데이터가 뛰어노는 AI 놀이터, 캐글

아이펠 강남 2기 김경현

2.5 평가지표의 최적화

■ 평가지표의 최적화 접근법

접근법	내용
간단하고 올바른 모델링 시행	- 평가지표가 RMSE나 로그 손실일 경우 모델의 **목적함수도 동일하게 지정**할 수 있음 - 별도의 처리를 하지 않아도 단순히 모델을 학습, 예측 시키는 것만으로 평가지표에 거의 최적화 됨
학습 데이터를 전처리하고 다른 평가지표를 최적화	- 평가지표가 RMSE일 경우, 주어진 학습 데이터의 목적함수의 로그를 취해 변환하고 목적함수를 RMSE로서 학습 시킨 뒤, 지수함수 값을 원래대로 변환하여 예측값을 제출하는 방법
다른 평가지표를 최적화하고 후처리	- 모델의 학습/예측을 수행한 후 평가지표의 성질에 근거하여 계산하거나, 최적화 알고리즘을 이용하여 임계값을 최적화하는 방법(2.5.2절, 2,6절 참조)
사용자 정의 목적함수를 사용	- 2.6.4절 참조
다른 평가지표를 최적화하고 학습 조기 종료	- 학습 조기 종료(Early stopping)의 평가 대상으로 최적화하고 싶은 평가지표를 설정 한 뒤, 해당 평가지표가 정확하게 최적화되는 시점에 학습을 멈추는 방법(4.1.3절 참조)

※ 출처 : [Coursera] How to Win a Data Science Competition: Learn from Top Kagglers

2.5 평가지표의 최적화

■ 임계값 최적화

○ 최적의 임계값을 구하는 방법

① 모든 임계값을 알아내는 방법

- 0.01부터 0.99까지 0.01씩 증가시키며 모든 임계값을 인수로 조사해 가장 좋은 점수가 되는 값을 채택

② 최적화 알고리즘을 이용하는 방법

- scipy.optimize 모듈 등을 이용해 '임계값을 인수로 점수를 반환하는 함수' 를 최적화

〈 넬더-미드(Nelder-Mead) 최적화 알고리즘〉

```python
# 임계값(threshold)의 최적화

from sklearn.metrics import f1_score
from sklearn.optimize import minimize

# 행 데이터 데이터 생성 준비
rand = np.random.RandomState(seed=71)
train_y_prob = np.linspace(0, 1.0, 10000)

# 실젯값과 예측값을 train_y, train_pred_prob 이었다고 가정
train_y = pd.Series(rand.uniform(0.0, 1.0, train_y_prob.size) < train_y_prob)
train_pred_prob = np.clip(train_y_prob * np.exp(rand.standard_normal(train_y_prob.shape) * 0.3), 0.0, 1.0)

# 임계값(thershold)을 0.5로 하면, F1은 0.722
init_threshold= 0.5
init_score = f1_score(train_y, train_pred_prob >= init_threshold)
print(init_threshold, init_score)

# 최적화의 목적함수를 설정
def f1_opt(x):
  return -f1_score(train_y, train_pred_prob >= x)

# scipy.optimize의 minimize 메서드에서 최적의 임곗값 구하기
# 구한 최적의 임곗값을 바탕으로 F1을 구하면 0.756이 된다
result = minimize(f1_opt, x0 = np.array([0.5]), method = "Nelder-Mead")
best_threshold = result['x'].item()
best_score = f1_score(train_y, train_pred_prob >= best_threshold)
print(best_threshold, best_score)
```

2.5 평가지표의 최적화

■ 임곗값 최적화와 OOF 예측의 필요성

○ OOF(Out-of-fold) 예측

- 데이터를 여러 개로 분할하여 그 중 하나를 예측 대상으로 삼고 나머지를 모델 학습 데이터로 삼아 변수를 예측하는 방법

○ F1-score를 최대로 하는 간단한 예는 OOF 예측으로 하지 않아도 큰 영향은 없지만, OOF로 최적화 수행 시 임계값 변화에 차이가 있거나 점수에 달라지는 현상을 확인할 수 있다. 복잡한 최적화를 수행할 때 과도한 평가 점수가 되는 걸 피할 수 있다는 장점이 있다.

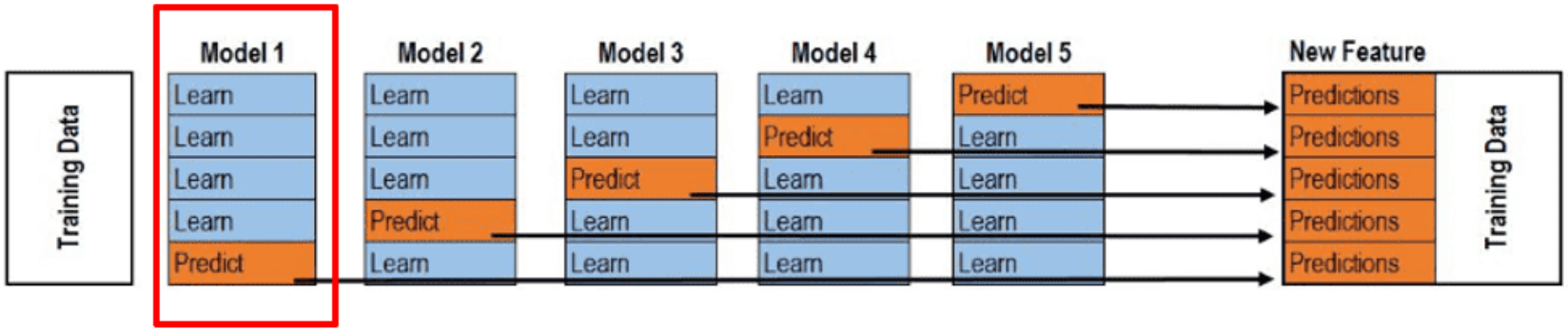
○ 실젯값과 예측 확률이 주어진 상황에서 OOF의 F1-score 최적화를 수행하는 순서

① 학습 데이터를 여러 개로 나눕니다. (여기서는 fold1, fold2, fold3. fold4)

② fold2, fold3, fold4의 실제값과 예측 확률에서 최적의 임계값을 구하고 그 임계값으로 fold1의 F1-score 계산한다.

③ 다른 fold들도 마찬가지로 자신 이외의 fold의 실제값과 예측 확률로부터 최적의 임계값을 구하고, 그 임계값으로 F1-score를 계산한다.

④ 테스트 데이터에 적용하는 임계값은 각 fold 임계값의 평균으로 한다.



참고자료 : [ML] Out of Fold(OOF) 방법으로 모델 평가하기

2.5 평가지표의 최적화

```python
import numpy as np
import pandas as pd

# ----------------------------------
# out-of-fold에서의 임곗값(threshold)의 최적화
# ----------------------------------
from scipy.optimize import minimize
from sklearn.metrics import f1_score
from sklearn.model_selection import KFold

# 샘플 데이터 생성 준비
rand = np.random.RandomState(seed=71)
train_y_prob = np.linspace(0, 1.0, 10000)

# 실젯값과 예측값을 다음과 같은 train_y, train_pred_prob이었다고 가정
train_y = pd.Series(rand.uniform(0.0, 1.0, train_y_prob.size) < train_y_prob)
train_pred_prob = np.clip(train_y_prob
* np.exp(rand.standard_normal(train_y_prob.shape) * 0.3), 0.0, 1.0)

# 교차 검증 구조로 임곗값을 구함
thresholds = []
scores_tr = []
scores_va = []

kf = KFold(n_splits=4, random_state=71, shuffle=True)
for i, (tr_idx, va_idx) in enumerate(kf.split(train_pred_prob)):
    tr_pred_prob, va_pred_prob = train_pred_prob[tr_idx], train_pred_prob[va_idx]
    tr_y, va_y = train_y.iloc[tr_idx], train_y.iloc[va_idx]

    # 최적화 목적함수를 설정
    def f1_opt(x):
        return -f1_score(tr_y, tr_pred_prob >= x)

    # 학습 데이터로 임곗값을 실시하고 검증 데이터로 평가를 수행
    result = minimize(f1_opt, x0=np.array([0.5]), method='Nelder-Mead')
    threshold = result['x'].item()
    score_tr = f1_score(tr_y, tr_pred_prob >= threshold)
    score_va = f1_score(va_y, va_pred_prob >= threshold)
    print(threshold, score_tr, score_va)

    thresholds.append(threshold)
    scores_tr.append(score_tr)
    scores_va.append(score_va)

# 각 fold의 임곗값 평균을 테스트 데이터에 적용
threshold_test = np.mean(thresholds)
print(threshold_test)

```

2.5 평가지표의 최적화

■ 예측 확률과 조정

1) 예측 확률의 왜곡

구분	내용
데이터가 충분하지 않은 경우	- 특히 데이터가 적을 때 0이나 1에 가까운 확률을 예측하기 어려움
모델 학습 알고리즘 상 타당한 확률을 예측하도록 최적화되지 않는 경우	- 모델이 로그 손실을 최소화하도록 학습할 경우, **충분한 데이터가 있으면 타당한 확률을 예측하지만 그렇지 않은 알고리즘에서는 예측 확률이 왜곡되기도 함** - GBDT, 신경망, 로지스틱 회귀일 때 분류 문제에서는 통상적인 설정으로 로그 손실을 목적함수로 삼아 학습하지만, 예를 들어 랜덤 포레스트에서는 다른 알고리즘으로 분류하므로 확률이 왜곡됨

2) 예측 확률의 조정

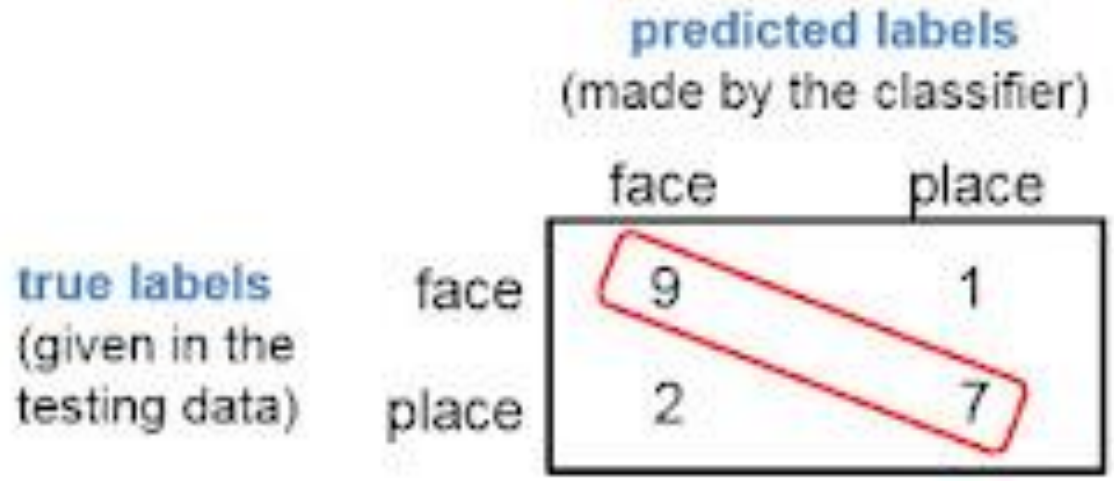
구분	내용
예측값을 n제곱	- 마지막에 **예측값을 n제곱(n은 0.9~1.1 정도)** 하는 경우, 확률을 충분히 학습하지 못했다고 판단하고 보정을 시도
0이나 1에 극단으로 가까운 확률은 회피	- 평가지표가 로그 손실일 경우, 큰 패널티를 피하는 등의 이유로 **출력 확률의 범위를 0.1~99.99%로 제한**하는 방법
스태킹(7.3절)	- 스태킹의 2계층 모델로 GBDT, 신경망, 로지스틱 회귀와 같이 타당한 확률을 예측하는 모델을 사용하는 방법 즉 ,**스태킹으로 최종 예측값을 출력할 경우 2계층에 이러한 모델을 사용하면 특별히 대응하지 않아도 확률이 보정됨**
CalibratedClassifierCV	- **사이킷럿 calibration 모듈의 CalibratedClassifierCV 클래스를 사용해 예측값을 보정**하는 방법 - 보정 수단으로 sigmoid 함수를 사용하는 플랫이나 등위 회귀를 선택할 수 있음

2.6 평가지표의 최적화 사례

■ 평가지표 최적화 방법(사례)

평가지표	유형	사례(경진대회)	최적화 방법
BA (Balanced Accuracy)	다중 클래스 분류	제1회 FR Frontier: 패션 이미지 의류 색상 분류	- 확률을 올바르게 예측한다는 가정 하에 **'확률 X 클래스 비율의 역수'가 최대인 클래스를 선택**
Mean-F1	다중 레이블 분류	Instacart Market Analysis	- 주문 ID별 각 상품의 구매 확률을 예측하고, 확률이 임계값 이상인 상품 ID를 예측값으로 출력 - 모든 주문ID에서 공통의 임계값을 사용하기보다는 **주문 ID별 최적 임계값을 구함**
QWK	다중 클래스 분류 (클래스 간 순서)	Prudential Life Insurance Assessment	- 연속값으로 출력한 후에 **클래스 간 임계값을 최적화** 하는 것 - 어느 값을 기준으로 클래스를 나눌지 그 구분값을 구하는 최적화 계산을 실시 - Scipy.optimize 모듈의 minimize 함수에서 넬더-미더 알고리즘 등을 사용
MAE	회귀	Allstate Claim Severity	- 사용자 정의 목적함수에서의 평가지표 근사에 따른 MAE 최적화 - MAE를 대신하는 근사함수를 목적함수로 사용(Fair 함수, Pseudo-Huber 함수)
MCC	이진분류	Bosch Production Line Performance	- MCC를 이용해 PR곡선을 살펴보면 곡선이 단조롭지 않은걸로 보아 최적의 임계값에서 MCC가 불안정하다고 예상할 수 있음 - PR-AUC에 따른 근사 및 모델 선택 - 로그 손실

별첨. BA 최적화 계산



regular ("overall") accuracy

$$\frac{9+7}{9+1+2+7} = 0.842$$

balanced accuracy

$$\left[\frac{9}{9+1} + \frac{7}{2+7}\right] / 2 = 0.839$$