Personalized-News-Recommendation(evaluator.py)

논문 요약

- **문제**: Contextual Bandit은 매 순간 한 개의 arm(콘텐츠)을 선택하고 그 보상만 관측됨 → 평가할 수 있는 정보가 제한됨
- **Live 테스트의 한계**: 실제 환경에서 새로운 알고리즘을 테스트하는 것은 **비용, 위험, 시간** 등 많은 제약이 있음
- **대안**: 과거 로그(logged) 데이터를 활용한 **off-policy 평가**
- **전제**:
  - 로그는 과거에 **무작위 정책(Uniform Random Policy)** 으로 수집되었음
  - 보상은 **선택된 arm 하나에 대해서만** 관측됨
- **해결책**: 알고리즘이 로그에 남아 있는 **같은 arm을 선택했을 때만 평가**에 반영 → 이 과정을 반복해서 성능 추정치를 얻음
- **장점**: 이 방법은 **편향되지 않은(unbiased)** 추정치를 제공함

> “내가 만든 추천 알고리즘이 진짜 잘 작동할까?”
>
> → 과거에 유저 A한테 뉴스 3번을 보여줬고 클릭했음.
>
> → 근데 내 알고리즘도 유저 A에게 뉴스 3번을 추천했을까?

코드 설명

전체 코드

```
import dataset
import random
import time


def evaluate(A, size=100, learn_ratio = 0.9):
    """
    Policy evaluator as described in the paper

    Parameters
    ----------
    A : class
        algorithm
    size : number
        Run the evaluation only on a portion of the dataset
    learn_ratio : number
        Perform learning(update parameters) only on a small portion of the traffic


    Returns
    -------
    learn : array
        contains the ctr for each trial for the learning bucket
    deploy : array
        contains the ctr for each trial for the deployment bucket
    """

    start = time.time()
    G_deploy = 0 # total payoff for the deployment bucket
    G_learn = 0  # total payoff for the learning bucket
    T_deploy = 1 # counter of valid events for the deployment bucket
    T_learn = 0  # counter of valid events for the learning bucket

    learn = []
    deploy = []
    if size == 100:
        events = dataset.events
    else:
```

```
    k = int(dataset.n_events * size / 100)
    events = random.sample(dataset.events, k)

  for t, event in enumerate(events):

    displayed = event[0]
    reward = event[1]
    user = event[2]
    pool_idx = event[3]

    chosen = A.choose_arm(G_learn + G_deploy, user, pool_idx)
    if chosen == displayed:
      if random.random() < learn_ratio:
        G_learn += event[1]
        T_learn += 1
        A.update(displayed, reward, user, pool_idx)
        learn.append(G_learn / T_learn)
      else:
        G_deploy += event[1]
        T_deploy += 1
        deploy.append(G_deploy / T_deploy)

  end = time.time()

  execution_time = round(end - start, 1)
  execution_time = (
    str(round(execution_time / 60, 1)) + "m"
    if execution_time > 60
    else str(execution_time) + "s"
  )
  print(
    "{:<20}{:<10}{}".format(
      A.algorithm, round(G_deploy / T_deploy, 4), execution_time
    )
  )

  return learn, deploy
```

```
import dataset
import random
import time
```

- `dataset` : 로그 데이터(`events`)를 가지고 있는 모듈
- `random` : 무작위 샘플링 및 확률 조건을 위한 표준 라이브러리
- `time` : 실행 시간 측정을 위해 사용

```
def evaluate(A, size=100, learn_ratio=0.9):
```

- `A` : 평가할 bandit 알고리즘 인스턴스 (`choose_arm` , `update` 메서드 필요함)
- `size` : 전체 데이터 중 몇 %를 사용할 것인지 (기본값: 100%)
- `learn_ratio` : 선택된 이벤트 중 학습에 사용할 비율 (기본값: 90%)

```
    start = time.time()
    G_deploy = 0
    G_learn = 0
    T_deploy = 1  # 0으로 하면 분모가 0이 될 수 있으므로 1로 시작
    T_learn = 0
```

- `G_` : 총 보상 (CTR의 분자 역할)
- `T_` : 이벤트 수 (CTR의 분모 역할)

```
    learn = []
    deploy = []
```

- 시간에 따른 CTR 변화를 기록하는 리스트
- 결과 그래프 등 시각화에 사용 가능

```
if size == 100:
    events = dataset.events
else:
    k = int(dataset.n_events * size / 100)
    events = random.sample(dataset.events, k)
```

- 전체 로그를 사용할지, 일부만 샘플링할지 결정
- `dataset.events` : `[displayed, reward, user, pool_idx]` 형식의 튜플 목록
  - `displayed` : 로그에 실제 표시된 arm
  - `reward` : 보상 (예: 클릭 여부 0 or 1)
  - `user` : 사용자 정보 (문맥 context)
  - `pool_idx` : 선택 가능한 arm들의 목록

```
for t, event in enumerate(events):

    displayed = event[0]  # 실제 보여준 뉴스
    reward = event[1]  # 유저가 클릭했는지 여부
    user = event[2]  # 유저 정보
    pool_idx = event[3] # 추천 후보 뉴스들
```

- 각 이벤트에서 필요한 정보
- Contextual Bandit에서는 이 정보들로 arm 선택 및 평가

```
    chosen = A.choose_arm(G_learn + G_deploy, user, pool_idx)
```

- 알고리즘이 현재 상태에서 arm을 선택하는 함수
- 입력으로 누적 보상, 사용자 정보, pool index 등을 받을 수 있음
- 논문에서 말하는 정책 π에 해당함.

```
        if chosen == displayed:
```

- 알고리즘이 실제 로그에서 선택된 arm과 동일한 arm을 골랐을 때만 평가 가능
- 즉, **replay 조건**: $\pi(h_{t-1}, x_t) = a_t$일 때만 보상 r_t을 사용할 수 있음

```
            if random.random() < learn_ratio:
                # 학습용 버킷
                G_learn += event[1]
                T_learn += 1
                A.update(displayed, reward, user, pool_idx)
                learn.append(G_learn / T_learn)
```

- reward와 이벤트 수 누적
- 알고리즘 내부 상태 업데이트
- 현재까지의 CTR(누적 보상 / 이벤트 수) 저장

```
            # 배포용 버킷
            else:
                G_deploy += event[1]
                T_deploy += 1
                deploy.append(G_deploy / T_deploy)
```

배포 버킷:

- 업데이트 없이 성능만 추적
- 보상과 이벤트 수 누적
- CTR 기록

```
    end = time.time()
```

실행 시간 측정 종료

```
execution_time = round(end - start, 1)
execution_time = (
    str(round(execution_time / 60, 1)) + "m"
    if execution_time > 60
    else str(execution_time) + "s"
)
```

- 실행 시간 포맷 설정 (60초 이상이면 분 단위로)

```
print(
    "{:<20}{:<10}{}".format(
        A.algorithm, round(G_deploy / T_deploy, 4), execution_time
    )
)
```

- 결과 출력: 알고리즘 이름, 배포 성능(CTR), 실행 시간
- `A.algorithm`은 알고리즘 객체가 가지고 있어야 할 속성 (str 타입)

```
return learn, deploy
```

- 학습 버킷과 배포 버킷의 **시간별 CTR 리스트** 반환
- 성능 추이 시각화나 비교에 활용 가능함.