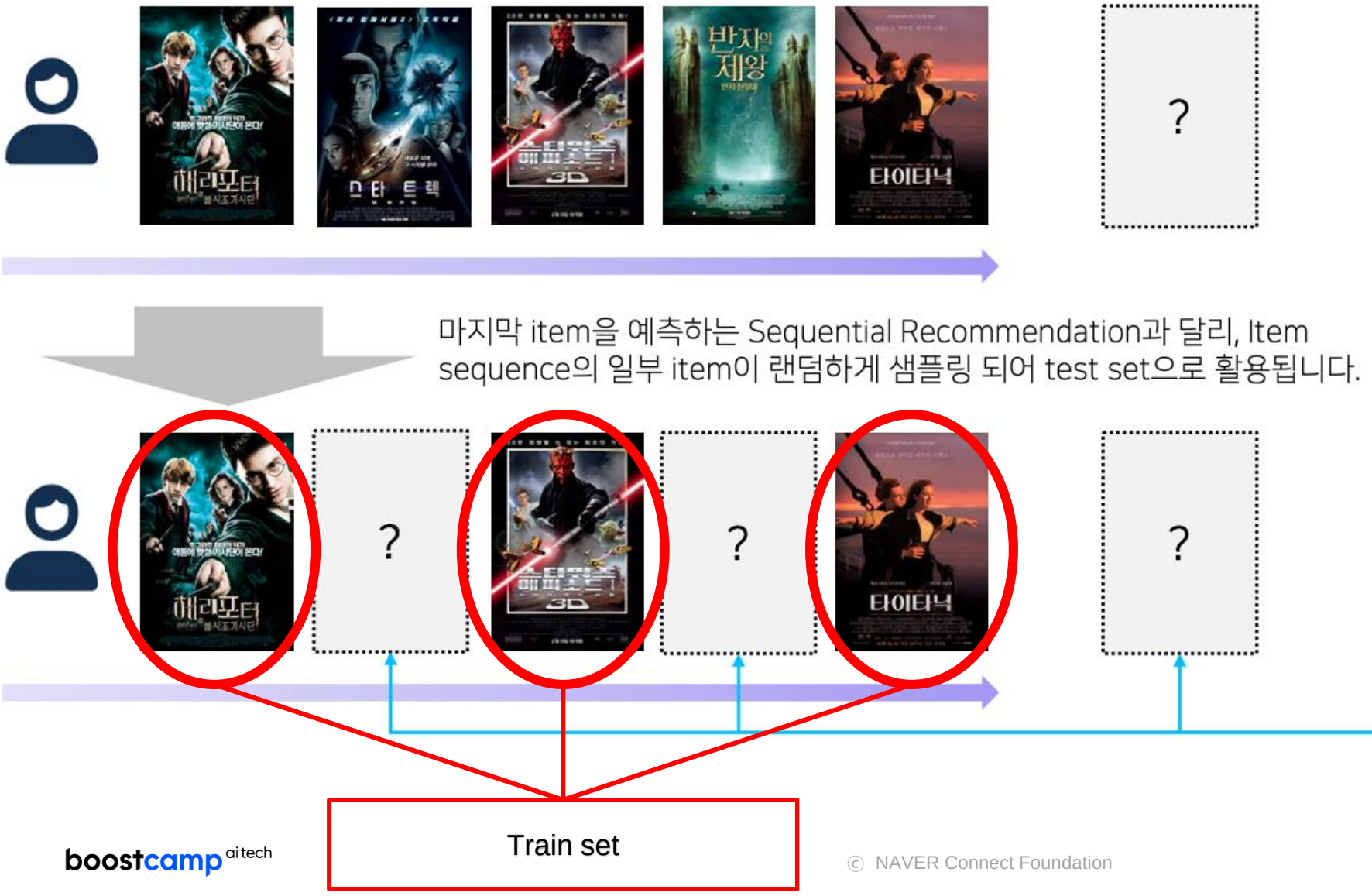
Movie Rec – train, test set

Concept document

우리의 TASK



USER_ID	ITEM_ID	TIMESTAMP
243	2904	1546559725
243	23985	1546560911
65	235	1546589911
124	54	1546588801

이게 정말 완벽한 데이터일까?

영화관에서 본

넷플릭스에서 본

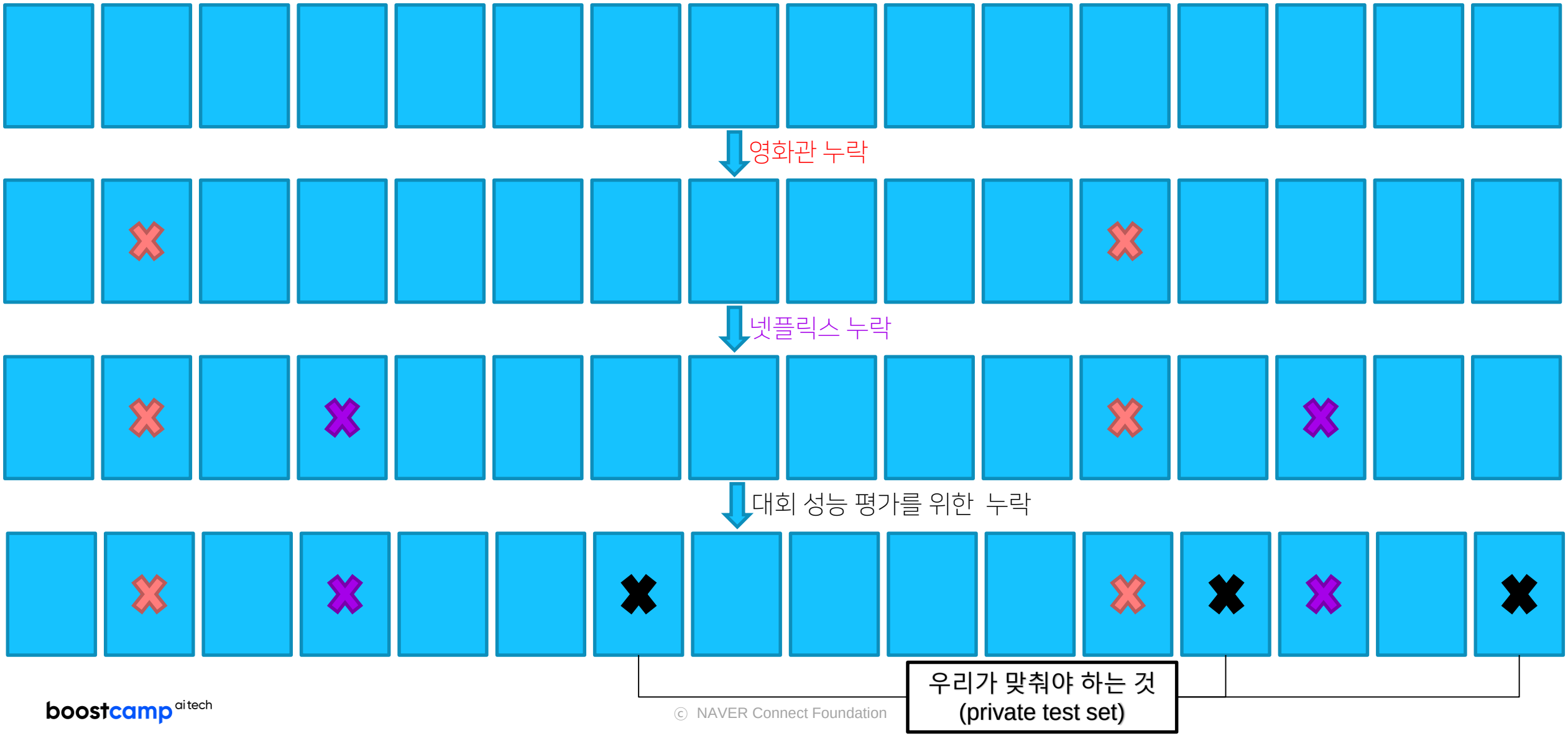
케이블에서 본

원본 데이터일지라도,
그 사람이 실제로 본 영화는
일부 누락되어 있다.

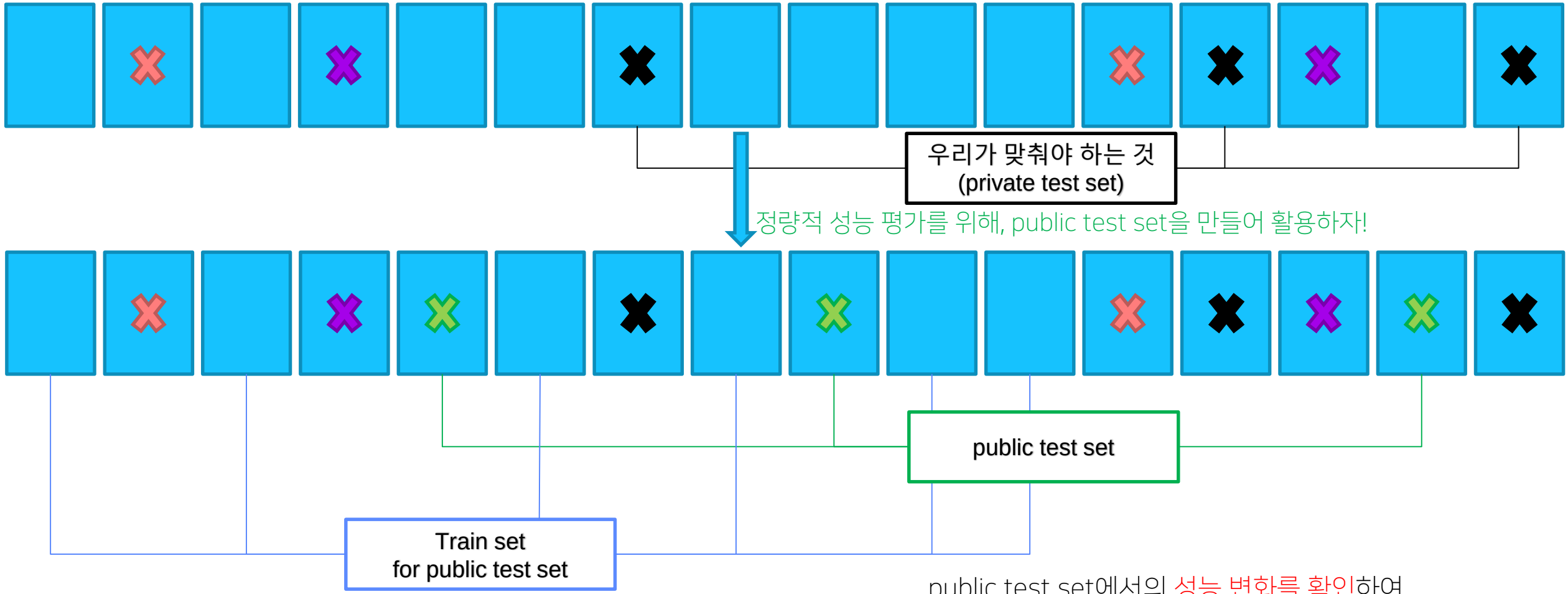
애당초 완벽한 데이터셋은 아니지만,
아무튼 맞춰 보자 !

naïve 한 접근이지만,
데이터의 패턴을 잘 학습했다면
어느 정도 맞출 수 있으리라 기대 가능

한번쯤 더 누락시켜도...



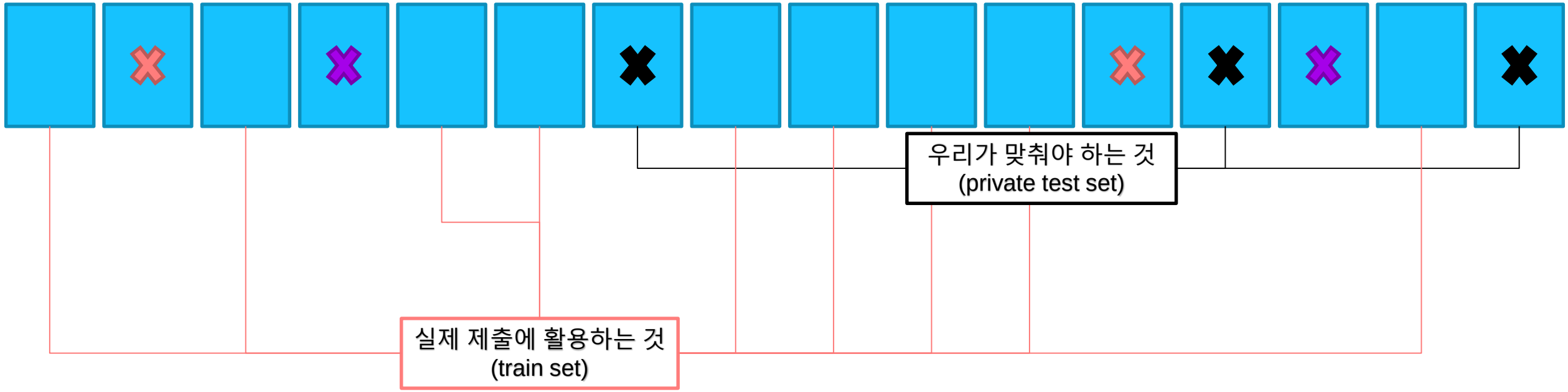
성능 개선에 대한 확실한 지표 !



public test set에서의 성능 변화를 확인하여,

모델 개선에 대한 정량화 된 지표로 활용 가능

제출 시 주의사항



앞선 결과에서 성능이 오른 방법들을 종합하여,

실제 train set에 똑같이 적용하여 제출

public test set 생성 기준

최대 10개 추출

표본이 적을 경우, 최대 25% 추출 (표본 4개당 1개)

파라미터 n_all로 제어

Sequential하게 2개, Static하게 8개

표본이 적을 경우, Sequential을 우선적으로 고려

파라미터 n_seq로 제어

mask를 이용하여 분리

```python
def generate_general_train_test_set(test_plays: pd.DataFrame, n_all=10, n_seq=2) -> Tuple[pd.DataFrame, pd.DataFrame]:
    np.random.seed(SEED)
    trains, labels = [], []
    for usr_id, tp in test_plays.groupby('user', as_index=False):
        _n_all = min(tp.shape[0]//4, n_all)
        _n_seq = min(_n_all, n_seq)
        _n_static = _n_all - _n_seq
        _n_all = _n_static + _n_seq

        _idxs = np.random.permutation(tp.shape[0]-_n_seq)[:_n_static]
        _mask = tp.index.isin(tp.index[_idxs])
        for i in range(_n_seq):
            _mask[-i-1] = True
        if VERBOSE:
            if _n_all != 10:
                print('_n_all:', _n_all)
                print(usr_id, _idxs)
                print(_n_static, _n_seq)

        trains.append(tp[~_mask])
        labels.append(tp[_mask])

    train_df = pd.concat(trains)
    label_df = pd.concat(labels)
    return train_df, label_df
```

Get Recall 구현 방법

```
%%time
recall_sum = 0
for user in tqdm(label_df['user'].unique()):
    preds = label_df[label_df['user'] == user]['item']
    labels = test_submission[test_submission['user'] == user]['item']

    recall_sum += preds.isin(labels).sum() / labels.shape[0]
```

```
100%|██████████| 31360/31360 [00:42<00:00, 737.54it/s]

CPU times: user 42.5 s, sys: 80 ms, total: 42.6 s
Wall time: 42.5 s
```

일반 for loop로 구현

약 43초 소요

```
async def _worker(user):
    preds = label_df[label_df['user'] == user]['item']
    labels = test_submission[test_submission['user'] == user]['item']

    return preds.isin(labels).sum() / labels.shape[0]
```

```
async def get_recall():
    loop = asyncio.get_event_loop()
    tasks = [loop.create_task(_worker(user)) for user in label_df['user'].unique()]
    result = await asyncio.gather(*tasks)
    print(sum(result) / label_df['user'].nunique())
    return sum(result) / label_df['user'].nunique()
```

```
start = int(time.time())
await get_recall()
print("***run time(sec) :", int(time.time()) - start)
```

```
0.09025510204079744
***run time(sec) : 43
```

비동기 방식으로 구현

약 43초 소요

(IO에 소요되는 시간 적음)

```
def _worker(user):
    preds = label_df[label_df['user'] == user]['item']
    labels = test_submission[test_submission['user'] == user]['item']

    return preds.isin(labels).sum() / labels.shape[0]
```

```
def get_recall():
    with Pool(os.cpu_count()) as p:
        result = p.map(_worker, label_df['user'].unique())
    return sum(result) / label_df['user'].nunique()
```

```
%%time
get_recall()
```

```
CPU times: user 104 ms, sys: 172 ms, total: 276 ms
Wall time: 5.66 s

0.09025510204079744
```

CPU 병렬 처리 방식으로 구현

약 6초 소요

(대략 CPU 코어 수 만큼 가속)