0717 ~ 0723 캐글 최종 모델

1등 솔루션

Then, we created a bunch of features in a brute-force way. For each combination of five raw categorical features (ip, os, app, channel, and device), we created the following click series-based feature sets (i.e., each feature set consists of 31 (=(2^5) - 1) features):

- click count within next one/six hours
- forward/backward click time delta
- average attributed ratio of past clicks

- click count within next one/six hours

  → 2. 시간 카운트
- forward/backward click time delta

  → 3.시간 델타
- average attributed ratio of past clicks

  → 4. 평균값 파생 피쳐

Next, we tried categorical feature embedding by using LDA/NMF/LSA. Here is the pseudo code to compute LDA topics of IPs related to app. (LDA is latent Dirichlet allocation)

```
apps_of_ip = {}
for sample in data_samples:
  apps_of_ip.setdefault(sample['ip'], []).append(str(sample['app']))
ips = list(apps_of_ip.keys())
apps_as_sentence = [' '.join(apps_of_ip[ip]) for ip in ips]
apps_as_matrix = CountTokenizer().fit_transform(apps_as_sentence)
topics_of_ips = LDA(n_components=5).fit_transform(apps_as_matrix)
```

We computed this feature for all the 20 (=5*(5-1)) combinations of 5 raw features and set the topic size to 5. This ended up with 100 new features. We also computed similar features using NMF and PCA, in total 300 new features. 0.9821 with a single LGB.

- " we tried categorical feature embedding by using LDA/NMF/LSA." → 5.LDA

데이터프레임 제작

- 기본적으로 다운샘플링, train_sample.csv(10만개짜리 데이터)에서는 피쳐를 추가할 때 성과가 오르지 않고, 오히려 뚝 떨어졌다.

  → 학습 데이터의 개수 부족으로 발생하는 문제라고 생각하고 전체 데이터 train을 가지고 피쳐엔지니어링을 하기로 결정.

I. 기본 제공 df

- 훈련 데이터
  - train_sample.csv
    - 행 10만 개
  - train.csv
    - 행 1억 개
- 테스트 데이터
  - test.csv
    - 행 1천만 개

II. 추가한 df

- full_train_no2_click&LDA_new.csv
  - full_train.no2_click_prev&next_new.csv(8.53GB) + full_train_no2_LDA.csv(77.8GB)
- full_test_no2_click&LDA.csv (25.9GB)
  - test_click_time_0.csv(6.51GB) + full_test_no2_LDA.csv (38.1GB)

  + 데이터 타입 압축

→ 724_make_data_full_click&LDA_train-Copy1.ipynb

---

1. Bagging
   - train[z] - z in range(5)
     - 랜덤시드를 다르게 해서 5개의 train.csv에서 1/5 비율로 모델 추출
   - 5개의 모델을 독립적으로 학습 후 output의 평균을 구하여 최종 답 도출

     → 현재 데이터 용량이 너무 크면서 생기는 kernel dead 현상 때문에 z=2 데이터만 사용

2. 시간 카운트
    - 1시간 이내
    - 6시간 이내
    - 현재 코드는 각각의 클릭 이벤트에 대해 1시간 앞뒤로 하는게 아니라, 14시면 14시에 groupby로 코딩됨

```
train[z]['real_clicks_1hr'] = train[z].groupby(['ip',
pd.Grouper(key='click_time', freq='1H')])
['click_time'].transform('count')
```

    - 두 가지 선택지
        - **ip만 가지고 유저** (clicks_hr_Mode) → **현재 이 버전**
        - 31가지 조합에 대해

3. 시간 델타
    - 가장 최근 델타 값 5개의 평균으로 구함
        - 최근에 있었던 과거 클릭(prev)
        - 직후에 나올 클릭(next)
    - 두 가지 선택지
        - ip만 가지고 유저 (clicks_ip_Mode)
        - **31가지 조합에 대해** (add_click_train, add_click_test)
            - prev만 처리한 df - **full_train.no{z}_click_prev.csv**
            - next만 처리한 df -
            - 합친 df - **full_train.no{z}_click_prev&next_new**
            - test - **full_train.no{z}_click_next.csv**

4. 평균값 파생 피쳐(att_ave_Mode) (예정)
    - 유저 조합에 대해 전체 평균을 타겟으로 (과거 대신 전체로)
    - 고민: 유저 정의가 디테일해지면 train과 test에서 많이 안 겹칠 수 있음
    - 아직 피쳐 추가 안함

5. LDA
    - **모든 조합에 대해(5 * 4 = 20) → 현재 이 버전**
        - LDA_Mode

```
train[z].to_csv(f'full_train_no{z}_LDA.csv', index=False)
tests[z].to_csv(f'full_test_no{z}_LDA.csv', index=False)
```

    - 아니면 IP에 대해? (4 + 4 = 8)
        - IP - x, x - IP