OTTO – Multi-Objective Recommender System

`Task overview`

The aim of this competition is to predict e-commerce clicks, cart additions, and orders. You'll build a multi-objective recommender system based on previous events in a user session.

Current recommender systems consist of various models with different approaches, ranging from simple matrix factorization to a transformer-type deep neural network. However, no single model exists that can simultaneously optimize multiple objectives. In this competition, you'll build a single entry to predict click-through, add-to-cart, and conversion rates based on previous same-session events.

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that ge
# You can also write temporary files to /kaggle/temp/, but they won't be saved
```

```
/kaggle/input/otto-recommender-system/sample_submission.csv
/kaggle/input/otto-recommender-system/test.jsonl
/kaggle/input/otto-recommender-system/train.jsonl
```

In [2]:
```
import pandas as pd
from pathlib import Path
import os
import random
import numpy as np
import json
from datetime import timedelta
from collections import Counter
from tqdm.notebook import tqdm
from heapq import nlargest

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()

import warnings
warnings.filterwarnings('ignore')

import math
```

In [3]:
```
Data_Path = Path('../input/otto-recommender-system')
Train_Path = Data_Path/'train.jsonl'
Test_Path = Data_Path/'test.jsonl'
Sample_sub_Path = Path('../input/otto-recommender-system/sample_submission.csv
```

In [4]:
```
with open(Train_Path, 'r') as f:
    print(f"We have {len(f.readlines()):,} lines in the training data")
```

```
We have 12,899,779 lines in the training data
```

In [5]:
```
sample_size = 150000

chunks = pd.read_json(Train_Path, lines=True, chunksize = sample_size)

for c in chunks:
    sample_train_df = c
    break
```

In [6]:
```
sample_train_df.set_index('session', drop=True, inplace=True)
sample_train_df.head()
```

Out[6]:

session	events
0	[{'aid': 1517085, 'ts': 1659304800025, 'type':...
1	[{'aid': 424964, 'ts': 1659304800025, 'type': ...
2	[{'aid': 763743, 'ts': 1659304800038, 'type': ...
3	[{'aid': 1425967, 'ts': 1659304800095, 'type':...
4	[{'aid': 613619, 'ts': 1659304800119, 'type': ...

Data structure

`session` *- the unique session id. Each session contains a list of time ordered events.*

`events` *- the time ordered sequence of events in the session. Each event contains 3 pieces of information:*

- `aid` *- the article id (product code) of the associated event*
- `ts` *- the Unix timestamp of the event (Unix time is the number of microseconds that have elapsed since 00:00:00 UTC on 1 January 1970)*
- `type` *- the event type, i.e., whether a product was clicked (`clicks`), added to the user's cart (`carts`), or ordered during the session (`orders`)*

In [7]:
```
example_session = sample_train_df.iloc[0].item()
print(f'This session was {len(example_session)} actions long \n')
print(f'The first action in the session: \n {example_session[0]} \n')

time_elapsed = example_session[-1]["ts"] - example_session[0]["ts"]
print(f'The first session elapsed: {str(timedelta(microseconds=time_elapsed))}

action_counts = {}
```

```
for action in example_session:
    action_counts[action['type']] = action_counts.get(action['type'], 0) + 1
print(f'The first session contains the following frequency of actions: {action_
```

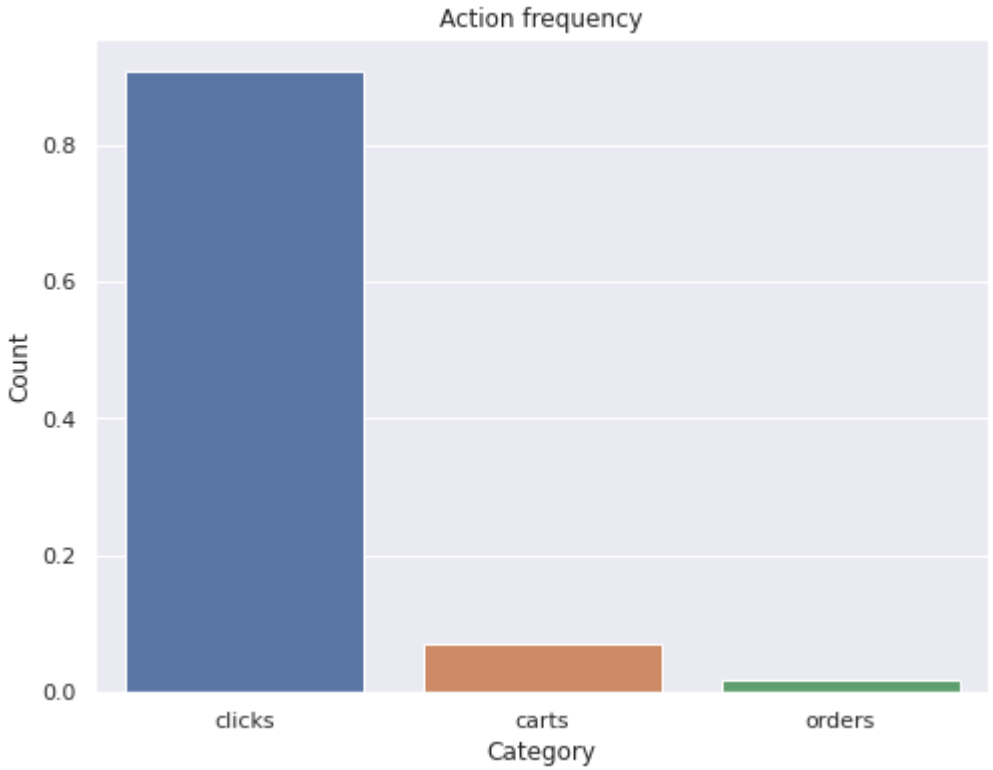
```
This session was 276 actions long

The first action in the session:
 {'aid': 1517085, 'ts': 1659304800025, 'type': 'clicks'}

The first session elapsed: 0:39:40.183682

The first session contains the following frequency of actions: {'clicks': 255,
'carts': 17, 'orders': 4}
```

Intital EDA

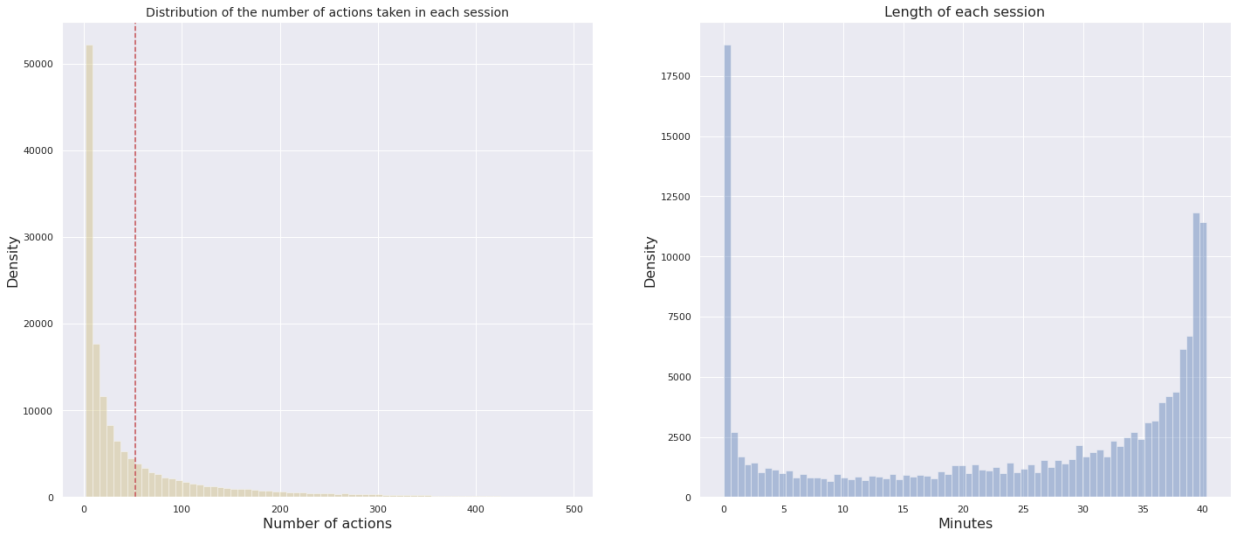
In [8]:
```
action_counts_list, article_id_counts_list, session_length_time_list, session_l
overall_action_counts = {}
overall_article_id_counts = {}

for i, row in tqdm(sample_train_df.iterrows(), total=len(sample_train_df)):

    actions = row['events']

    action_counts = {}
    article_id_counts = {}
    for action in actions:
        action_counts[action['type']] = action_counts.get(action['type'], 0) +
        article_id_counts[action['aid']] = article_id_counts.get(action['aid']
        overall_action_counts[action['type']] = overall_action_counts.get(actio
        overall_article_id_counts[action['aid']] = overall_article_id_counts.ge

    session_length_time = actions[-1]['ts'] - actions[0]['ts']

    action_counts_list.append(action_counts)
    article_id_counts_list.append(article_id_counts)
    session_length_time_list.append(session_length_time)
    session_length_action_list.append(len(actions))

sample_train_df['action_counts'] = action_counts_list
sample_train_df['article_id_counts'] = article_id_counts_list
sample_train_df['session_length_unix'] = session_length_time_list
sample_train_df['session_length_minutes'] = sample_train_df['session_length_uni
sample_train_df['session_length_action'] = session_length_action_list
```

```
100% ████████████████████████████████████████ 150000/150000 [00:23<00:00, 6837.68it/s]
```

In [9]:
```
total_actions = sum(overall_action_counts.values())

plt.figure(figsize=(8,6))
sns.barplot(x=list(overall_action_counts.keys()), y=[i/total_actions for i in o
plt.title(f'Action frequency', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('Category', fontsize=12)
plt.show()
```



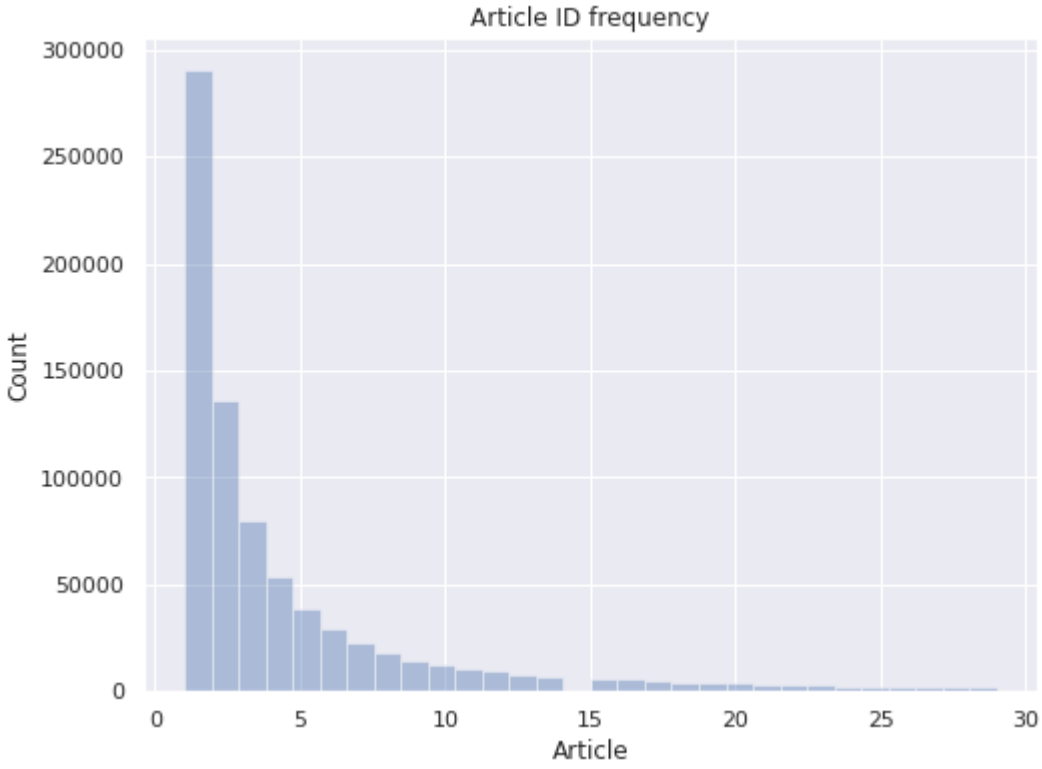
In [10]:
```
fig, ax = plt.subplots(1,2, figsize=(24, 10))

p = sns.distplot(sample_train_df['session_length_action'], color="y", bins= 70
p.set_xlabel("Number of actions", fontsize = 16)
p.set_ylabel("Density", fontsize = 16)
p.set_title("Distribution of the number of actions taken in each session", font
p.axvline(sample_train_df['session_length_action'].mean(), color='r', linestyle

p = sns.distplot(sample_train_df['session_length_minutes'], color="b", bins= 70
p.set_xlabel("Minutes", fontsize = 16)
p.set_ylabel("Density", fontsize = 16)
p.set_title("Length of each session", fontsize = 16);
```



In [11]:
```
print(f'{round(len(sample_train_df[sample_train_df["session_length_action"]<10]
```

```
34.8% of the sessions had less than 10 actions
```

In [12]:
```
article_id_freq = list(overall_article_id_counts.values())
cut_off = [i for i in article_id_freq if i<30]

plt.figure(figsize=(8,6))
sns.distplot(cut_off, bins=30, kde=False);
plt.title(f'Article ID frequency', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('Article', fontsize=12);
```



In [13]:
```
print(f'Frequency of most common articles: {sorted(list(overall_article_id_cou
res = nlargest(5, overall_article_id_counts, key = overall_article_id_counts.g
print(f'IDs for those common articles: {res}')
```

```
Frequency of most common articles: [3877, 4259, 4323, 4503, 5000]

IDs for those common articles: [29735, 1603001, 832192, 1733943, 108125]
```

Baseline

The test data contains truncated session data similar to that of the training data. The task is to predict the next aid clicked after the session truncation, as well as the the remaining aids that are added to carts and orders; you may predict up to 20 values for each session type

Submissions are evaluated on Recall each action type, and the three recall values are weight-averaged: {'clicks': 0.10, 'carts': 0.30, 'orders': 0.60} . It is important to get the 'orders' predictions correct as they carry most of the weigthing

For each session in the test data, your task it to predict the aid values for each type that occur after the last timestamp ts the test session. In other words, the test data contains sessions truncated by timestamp, and you are to predict what occurs after the point of truncation.

For clicks there is only a single ground truth value for each session, which is the next aid clicked during the session (although you can still predict up to 20 aid values). The ground truth for carts and orders contains all aid values that were added to a cart and ordered respectively during the session.

_Each session and type combination should appear on its own session_type row in the submission (3 rows per session), and predictions should be space delimited. This can be seen in the sample_test*df below.*

In [14]:
```
with open(Test_Path, 'r') as f:
    print(f"We have {len(f.readlines()):,} lines in the test data")
```

```
We have 1,671,803 lines in the test data
```

In [15]:
```
sample_size = 150

chunks = pd.read_json(Test_Path, lines=True, chunksize = sample_size)

for c in chunks:
    sample_test_df = c
    break
```

In [16]:
```
sample_test_df.head()
```

Out[16]:

	session	events
0	12899779	[{'aid': 59625, 'ts': 1661724000278, 'type': '...
1	12899780	[{'aid': 1142000, 'ts': 1661724000378, 'type':...
2	12899781	[{'aid': 141736, 'ts': 1661724000559, 'type': ...
3	12899782	[{'aid': 1669402, 'ts': 1661724000568, 'type':...
4	12899783	[{'aid': 255297, 'ts': 1661724000572, 'type': ...

Below shows a sample submission. For each session in the test set there is a prediction (labels). This predicts what articles will be next interacted with in that session. For each session there are three actions (clicks, carts, orders), predictions are made for all three actions.

In [17]:
```
sample_submission = pd.read_csv(Sample_sub_Path)
sample_submission.head()
```

Out[17]:

	session_type	labels
0	12899779_clicks	129004 126836 118524
1	12899779_carts	129004 126836 118524
2	12899779_orders	129004 126836 118524
3	12899780_clicks	129004 126836 118524
4	12899780_carts	129004 126836 118524

Lets find the most common article for each different type of action.

In [18]:
```
sample_size = 150000

chunks = pd.read_json(Train_Path, lines=True, chunksize = sample_size)

clicks_article_list = []
carts_article_list = []
orders_article_list = []

for e, c in enumerate(chunks):

    if e > 2:
        break

    sample_train_df = c

    for i, row in c.iterrows():
        actions = row['events']
        for action in actions:
            if action['type'] == 'clicks':
                clicks_article_list.append(action['aid'])
            elif action['type'] == 'carts':
                carts_article_list.append(action['aid'])
            else:
                orders_article_list.append(action['aid'])
```

In [19]:
```
article_click_freq = Counter(clicks_article_list)
article_carts_freq = Counter(carts_article_list)
article_order_freq = Counter(orders_article_list)
```

In [20]:
```
top_click_article = nlargest(20, article_click_freq, key = article_click_freq.
top_carts_article = nlargest(20, article_carts_freq, key = article_carts_freq.
top_order_article = nlargest(20, article_order_freq, key = article_order_freq.
```

In [21]:
```
frequent_articles = {'clicks': top_click_article, 'carts':top_carts_article, '
```

In [22]:
```
for action in ['clicks', 'carts', 'order']:
    print(f'Most frequent articles for {action}: {frequent_articles[action][-5
```

```
Most frequent articles for clicks: [1196256, 1502122, 673407, 508883, 1743151]
Most frequent articles for carts: [1743151, 544144, 1498443, 351335, 247240]
Most frequent articles for order: [923948, 801774, 247240, 1111967, 563117]
```

There is some overlap but the articles do change for the different actions!

This baseline will use the fact that people will often interact with articles they have previouslt interacted with. The prediction will consist of the top 20 most frequent articles in the session. If there are less than 20 articles in the session the prediction will be padded with the most frequent articles in the training data as found above.

In [23]:
```
test_data = pd.read_json(Test_Path, lines=True, chunksize=1000)

preds = []

for chunk in tqdm(test_data, total=1671):

    for i, row in chunk.iterrows():
        actions = row['events']
```

```
        article_id_list = []
        for action in actions:
            article_id_list.append(action['aid'])

        article_freq = Counter(article_id_list)
        top_articles = nlargest(20, article_freq, key = article_freq.get)

        padding_size = -(20 - len(top_articles))
        for action in ['clicks', 'carts', 'order']:
            top_articles = top_articles + frequent_articles[action][padding_si
            preds.append(" ".join([str(id) for id in top_articles]))
```


1672/? [04:07<00:00, 7.90it/s]

In [24]:
```
sample_submission['labels'] = preds
```

In [25]:
```
sample_submission.to_csv('submission.csv', index=False)
```

Thank You