GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

EVENT RECOMMENDATION SYSTEM

A Report for the Evaluation 3 of Project 2

Submitted by

SHOURYA MEHROTRA

(1613105117/16SCSE105070)

in partial fulfillment for the award of the degree of

BACHELOR IN TECHNOLOGY

IN

COMPUTER SCIENCE

(WITH SPECIALIZATION IN CLOUD COMPUTING AND VIRTUALIZATION)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Under the Supervision of Supervisor DR. D. NAGESHWARA RAO

MAY 2020

TABLE OF CONTENTS

ABSTRACT

A recommender system is a discovery system. The system learns from the data and provides recommendations to users. Without the user specifically searching for that item, that item was brought automatically by the system.It sounds like magic.And this magic has been used by Amazon and Netflix since decades ago.How awesome it is, when you open Spotify and it already gives you a list of songs to listen to.A recommendation system is a type of information filter, which can learn users' interests and hobbies according to their profile or historical behaviours, and then predict their ratings or preferences for a given item. It changes the way businesses communicate and strengthens the interactivity between user and provider. With the large number of events published all the time in event-based social networks (EBSN), it has become increasingly difficult for users to find the events that best match their preferences. Recommender systems appear as a natural solution to the entire problem . However, the event recommendation scenario is quite different from typical recommendation domains (e.g. movies), since there is an intrinsic new item problem involved (i.e. events can not be "consumed" before their occurrence) and scarce collaborative information. Although some few works have appeared in this area, there is still lacking in the literature an extensive analysis of the different characteristics of EBSN data that can affect the design of event recommenders. We examine the topic of unseen item recommendation through a user study of academic (scientific) talk recommendation, where we aim to correctly estimate a ranking function for each user, predicting which talks would be of most interest to them.

INTRODUCTION

A recommendation system helps users find compelling content in large corpora. For example, the Google Play Store provides millions of apps, while YouTube provides billions of videos. More apps and videos are added every day. One can use search to access content. However, a recommendation engine can display items that users might not have thought to search for on their own.

Recommendation engines have been around for a while and there have been some key learnings to leverage:

- 1. A user's actions are the best indicator of user intent. Ratings and feedback tends to be very biased and lower volumes.
- 2. Past actions and purchases drive new purchases and the overlap with other people's purchases and actions is a fantastic predictor.

In any such analysis where all the inputs are behavioural which generally are because through survey reports or feedback people would not want to give the right verdict while through search or clicks one can understand how well a user likes something or the kind of things he or she likes and the kind of categories that do interest.

Every single tie a user does some activity to the website or application data can be recorded, either through a websites own analytical system or through some software which help a lot in establishing digital marketing as one of the key ways to grow a business by understanding what exactly a user wants and who are the users which want you products.

And this can give birth to an idea of event marketing –

Event marketing is defined by the tools, techniques and channels you use to promote an event to an audience, usually with the hope of getting them to buy tickets or attend.

Event marketing begins with launching an event idea, through to persuading attendees to invite their friends or colleagues and attracting a steady pipeline of leads through channels such as email marketing, blogging and advertising.

B2B (business to business) event marketing is selling an event to other businesses in order to get them to attend, sponsor or exhibit. The channels of advertising may be the same (for example social media, email marketing, PPC) but the tone of voice and USPs may be different. For example at a consumer event, the sales techniques will be activated towards personal interest and benefit, whereas for a business event the rewards may need to be more tangible such as helping a business to increase revenue, generate leads or gain a competitive advantage.

In-depth about recommender system

In general term, there are two kinds of recommender system known by us, a human. Well, not all human.

1. Content-based filtering

The type of recommender system that can easily be digested by our brain. Without a sign of short-circuiting or exploding.For example, you are an avid novel reader. And you like “And then there were none” by Agatha Christie. You bought it from an online bookstore.It makes sense if the bookstore will show you “The ABC Murders” the next time you open the website.Why? Because both of them written by Agatha Christie.Hence, the Content-based filtering model will recommend you that title.While Content-based filtering is easily digested by our brain and looks so simple, it can fail to guess the real behavior of the user.For example, I don’t like Hercule Poirot, but I like other detectives in her novels. In that case, “The ABC Murders” should not be recommended for me.

2. Collaborative filtering

This type will overcome the previous problem. Essentially, the system record all the previous interaction of the user on the website. And provide recommendations based on that.How does it work?Take a look at this scenario.

There are two users, A and B.

A bought item 1

A bought item 2

A bought item 3

B bought item 1

B bought item 3

The collaborative filtering **will recommend B item 2** since there is another user who bought items 1 and 3 also bought item 2.You might say, wow, they could be sporadically bought together in coincidence.But, what if, there are 100 users who have the same behavior with user A.That was, the so-called, **the power of crowds**.So, why waiting. Let's just start creating Collaborative filtering system in your production environment.While it has an extremely good performance. It has several serious issues. More importantly when you are trying to create a production-ready system.

EXISTING SYSTEM

The downside of Collaborative filtering

1. It doesn’t know about context. In contrast with Content-based filtering that recommends similar items, Collaborative filtering will not recommend based on similarity. When this is your concern, the solution is going hybrid. Combine both methods.

2. It needs huge hardware resources since you need to store a user-item matrix. Imagine if you open your e-commerce website and it has 100K users. At the same time, you serve 10K products. In this case, you will need 10K x 100K matrix with each element hold 4 bytes integer. Yep, you need 4GB memory just for storing the matrix. Not even doing other things.

3. Cold start. A new user will not get any benefit from the system since you have no idea about him.

4. The unchangeable. If you are not doing anything on the website, the result of the recommender system will stay the same. The user will think that there is nothing new on the website. And they will leave.

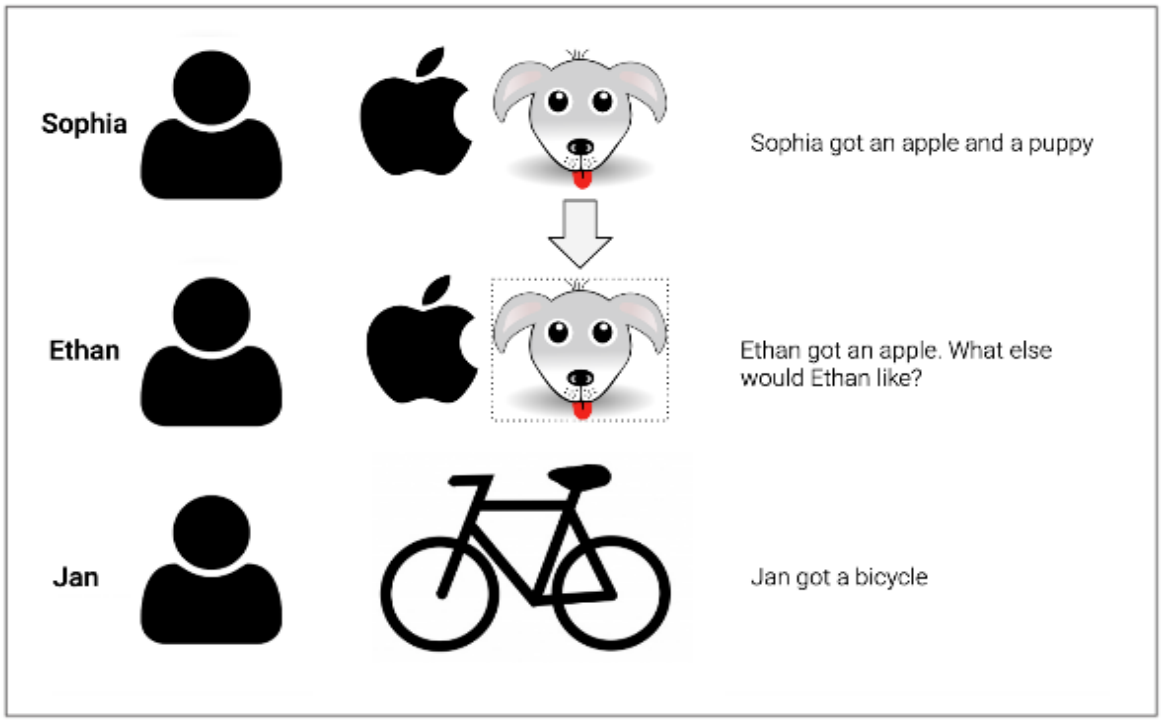
Solution

1. Batch computation on the general recommendation indicator.

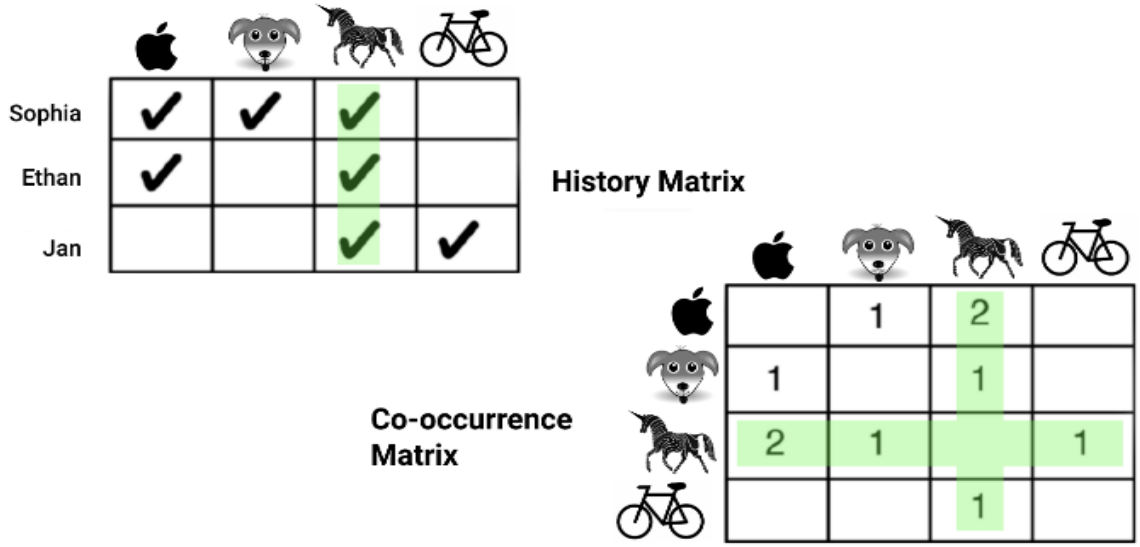
2. Query on real-time, without using the user-item matrix, but take several latest interactions of the user and query it to the system.

PROPOSED MODEL

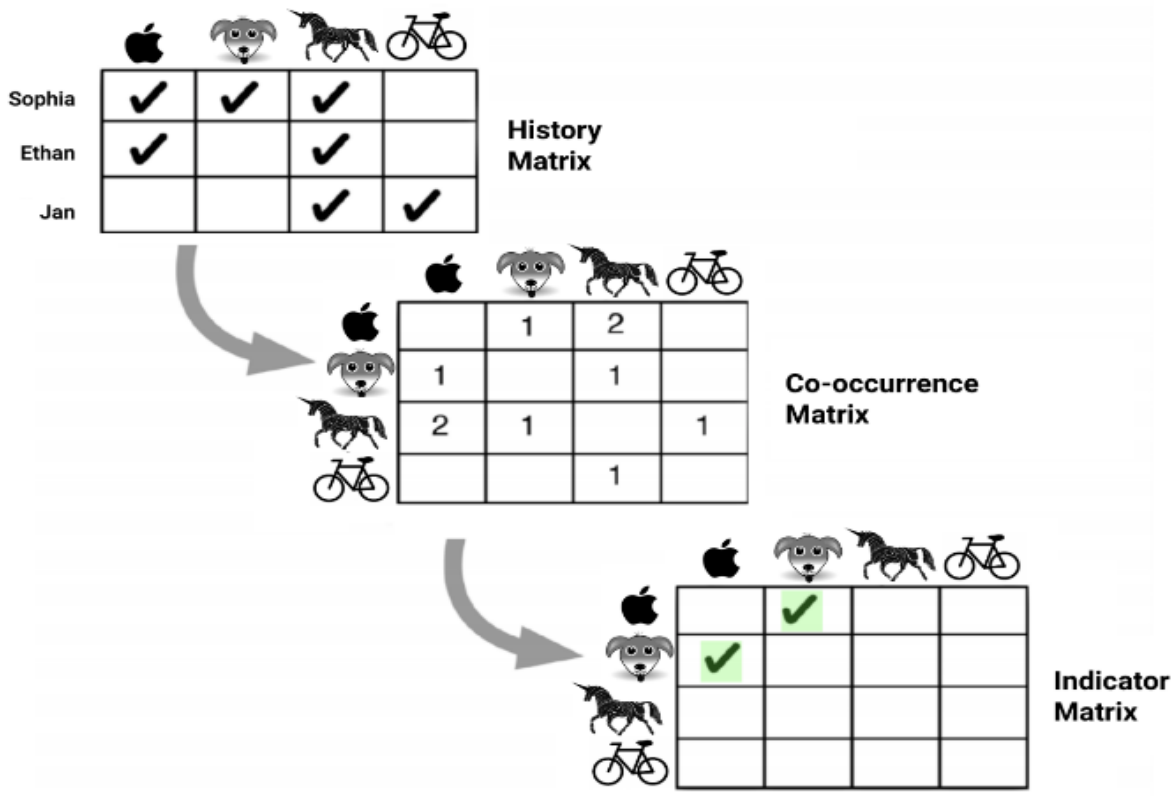
Recommendation systems generally look for overlap or co-occurrence to make a recommendation.Like in the following example where we recommend Ethan a puppy based on a similarity of Ethan with Sophia:



In practise, a recommendation engine computes a co-occurrence matrix from a history matrix of events and actions. This is simple enough but there are challenges to overcome in real world scenarios. What if everyone wants a unicorn.Does the high co-occurrence of unicorns in the following example make a good recommendation.



After the recommendation system has computed the co-occurrence matrix we have to apply statistics to filter out the sufficiently anomalous signals to be interesting as a recommendation.



IMPLEMENTATION

Recommender system is using Python.

- Here the code is implemented in Jupyter Notebook.
- Elasticsearch is also used, it is an open-source search engine, that can enable you to search your document really fast.
- Postman is also used which is an API development tool. It is needed to simulate the query into elasticsearch. As elasticsearch can be accessed via http.

```
for item in the_bag:
print(item)


pip install numpy
pip install scipy
pip install pandas
pip install jupyter
pip install requests


import pandas as pd
import numpy as np


df = pd.read_csv('events.csv')
df.shape


df.head()


df.event.unique()


trans = df[df['event'] == 'transaction']
trans.shape
visitors = trans['visitorid'].unique()
items = trans['itemid'].unique()print(visitors.shape)
print(items.shape)


trans2 = trans.groupby(['visitorid']).head(50)
trans2.shape
trans2['visitors'] = trans2['visitorid'].apply(lambda x : np.argwhere(visitors == x)[0][0])
trans2['items'] = trans2['itemid'].apply(lambda x : np.argwhere(items == x)[0][0])trans2
```

```
from scipy.sparse import csr_matrix


occurences = csr_matrix((visitors.shape[0], items.shape[0]), dtype='int8')def
set_occurences(visitor, item):
    occurences[visitor, item] += 1trans2.apply(lambda row: set_occurences(row['visitors'],
row['items']), axis=1)occurences


cooc = occurences.transpose().dot(occurences)
cooc.setdiag(0)


def xLogX(x):
    return x * np.log(x) if x != 0 else 0.0def entropy(x1, x2=0, x3=0, x4=0):
    return xLogX(x1 + x2 + x3 + x4) - xLogX(x1) - xLogX(x2) - xLogX(x3) - xLogX(x4)def
LLR(k11, k12, k21, k22):
    rowEntropy = entropy(k11 + k12, k21 + k22)
    columnEntropy = entropy(k11 + k21, k12 + k22)
    matrixEntropy = entropy(k11, k12, k21, k22)
    if rowEntropy + columnEntropy < matrixEntropy:
        return 0.0
    return 2.0 * (rowEntropy + columnEntropy - matrixEntropy)def rootLLR(k11, k12, k21,
k22):
    llr = LLR(k11, k12, k21, k22)
    sqrt = np.sqrt(llr)
    if k11 * 1.0 / (k11 + k12) < k21 * 1.0 / (k21 + k22):
        sqrt = -sqrt
    return sqrt




row_sum = np.sum(cooc, axis=0).A.flatten()
column_sum = np.sum(cooc, axis=1).A.flatten()
total = np.sum(row_sum, axis=0)pp_score = csr_matrix((cooc.shape[0], cooc.shape[1]),
dtype='double')
cx = cooc.tocoo()
for i,j,v in zip(cx.row, cx.col, cx.data):
    if v != 0:
        k11 = v
        k12 = row_sum[i] - k11
        k21 = column_sum[j] - k11
        k22 = total - k11 - k12 - k21
        pp_score[i,j] = rootLLR(k11, k12, k21, k22)


result = np.flip(np.sort(pp_score.A, axis=1), axis=1)
result_indices = np.flip(np.argsort(pp_score.A, axis=1), axis=1)
```

result[8456]

array([15.33511076, 14.60017668, 3.62091635, ..., 0. ,
0. , 0.])

result_indices[8456]

array([8682, 380, 8501, ..., 8010, 8009, 0], dtype=int64)

```
minLLR = 5
indicators = result[:, :50]
indicators[indicators < minLLR] = 0.0indicators_indices = result_indices[:,
:50]max_indicator_indices = (indicators==0).argmax(axis=1)
max = max_indicator_indices.max()indicators = indicators[:, :max+1]
indicators_indices = indicators_indices[:, :max+1]


import requests
import json


actions = []
for i in range(indicators.shape[0]):
    length = indicators[i].nonzero()[0].shape[0]
    real_indicators = items[indicators_indices[i, :length]].astype("int").tolist()
    id = items[i]

    action = { "index" : { "_index" : "items2", "_id" : str(id) } }

    data = {
        "id": int(id),
        "indicators": real_indicators
    }

    actions.append(json.dumps(action))
    actions.append(json.dumps(data))

    if len(actions) == 200:
        actions_string = "\n".join(actions) + "\n"
        actions = []

        url = "http://127.0.0.1:9200/_bulk/"
        headers = {
            "Content-Type" : "application/x-ndjson"
        }
        requests.post(url, headers=headers, data=actions_string)if len(actions) > 0:
    actions_string = "\n".join(actions) + "\n"
    actions = []    url = "http://127.0.0.1:9200/_bulk/"
```

```
    headers = {
        "Content-Type" : "application/x-ndjson"
    }
    requests.post(url, headers=headers, data=actions_string)
```

```
{
"count": 12025,
"_shards": {
"total": 1,
"successful": 1,
"skipped": 0,
"failed": 0
}
}
```

```
{
"id": 240708,
"indicators": [
305675,
346067,
312728
]
}
```

```
{
"query": {
"bool": {
"should": [
{ "terms": {"indicators" : [240708], "boost": 2}}
]
}
}
}
```

```
{
"query": {
"bool": {
"should": [
{ "terms": {"indicators" : [240708]}},
{ "constant_score": {"filter" : {"match_all": {}}, "boost" : 0.000001}}
]
}
}
}
```

```
popular = np.zeros(items.shape[0])def inc_popular(index):
    popular[index] += 1trans2.apply(lambda row: inc_popular(row['items']), axis=1)
```

```
actions = []
for i in range(indicators.shape[0]):
    length = indicators[i].nonzero()[0].shape[0]
    real_indicators = items[indicators_indices[i, :length]].astype("int").tolist()
    id = items[i]

    action = { "index" : { "_index" : "items3", "_id" : str(id) } }

#     url = "http://127.0.0.1:9200/items/_create/" + str(id)
    data = {
        "id": int(id),
        "indicators": real_indicators,
        "popular": popular[i]
    }

    actions.append(json.dumps(action))
    actions.append(json.dumps(data))

    if len(actions) == 200:
        actions_string = "\n".join(actions) + "\n"
        actions = []

        url = "http://127.0.0.1:9200/_bulk/"
        headers = {
            "Content-Type" : "application/x-ndjson"
        }
        requests.post(url, headers=headers, data=actions_string)if len(actions) > 0:
    actions_string = "\n".join(actions) + "\n"
    actions = []url = "http://127.0.0.1:9200/_bulk/"
    headers = {
        "Content-Type" : "application/x-ndjson"
    }
    requests.post(url, headers=headers, data=actions_string)


{
"id": 240708,
"indicators": [
305675,
346067,
312728
],
"popular": 3.0
}


{
"query": {
"function_score":{
"query": {
"bool": {
```

```
"should": [
{ "terms": {"indicators" : [240708], "boost": 2}},
{ "constant_score": {"filter" : {"match_all": {}}, "boost" : 0.000001}}
]
}
},
"functions":[
{
"filter": {"range": {"popular": {"gt": 0}}},
"script_score" : {
"script" : {
"source": "doc['popular'].value * 0.1"
}
}
}
],
"score_mode": "sum",
"min_score" : 0
}
}
}{
"query": {
"function_score":{
"query": {
"bool": {
"should": [
{ "terms": {"indicators" : [240708], "boost": 2}},
{ "constant_score": {"filter" : {"match_all": {}}, "boost" : 0.000001}}
]
}
},
"functions":[
{
"filter": {"range": {"popular": {"gt": 1}}},
"script_score" : {
"script" : {
"source": "0.1 * Math.log(doc['popular'].value)"
}
}
},
{
"filter": {"match_all": {}},
"random_score": {}
}
],
"score_mode": "sum",
"min_score" : 0}
}
}
```

OUTPUT AND SCREENSHOTS

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
2	1433221999827	111016	view	318965	NaN
3	1433221955914	483717	view	253185	NaN
4	1433221337106	951259	view	367447	NaN

- It has five columns.

1. Timestamp, the timestamp of the event.
2. Visitorid, the id of the user
3. Itemid, the id of the item
4. Event, the event
5. Transactionid, an id of the transaction if the event is a transaction

- After which this is used to unique identify the required events:

```
df.event.unique()
```

	timestamp	visitorid	event	itemid	transactionid	visitors	items
0	1433222276276	599528	transaction	356475	4000.0	0	0
1	1433193500981	121688	transaction	15335	11117.0	1	1
2	1433193915008	552148	transaction	81345	5444.0	2	2
3	1433176736375	102019	transaction	150318	13556.0	3	3
4	1433174518180	189384	transaction	310791	7244.0	4	4
5	1433184261340	350566	transaction	54058	8796.0	5	5
6	1433184261371	350566	transaction	284871	8796.0	5	6
7	1433182772710	404403	transaction	150100	5216.0	6	7
8	1433164275232	505565	transaction	243566	11713.0	7	8
9	1433180403455	945184	transaction	245400	2415.0	8	9
10	1433177808042	1406787	transaction	336832	15915.0	9	10

- The next step: Create the user-item matrix

- Once on applying the code we can see in the final output as follows.


```
{
    "_index": "items3",
    "_type": "_doc",
    "_id": "461686",
    "_score": 1.3E-5,
    "_source": {
        "id": 461686,
        "indicators": [
            218794,
            171878,
            10572,
            32581,
            124081,
            267491,
            357529,
            108924
        ],
        "popular": 130.0
    }
},
```

CONCLUSION AND FUTURE ENHANCEMENTS

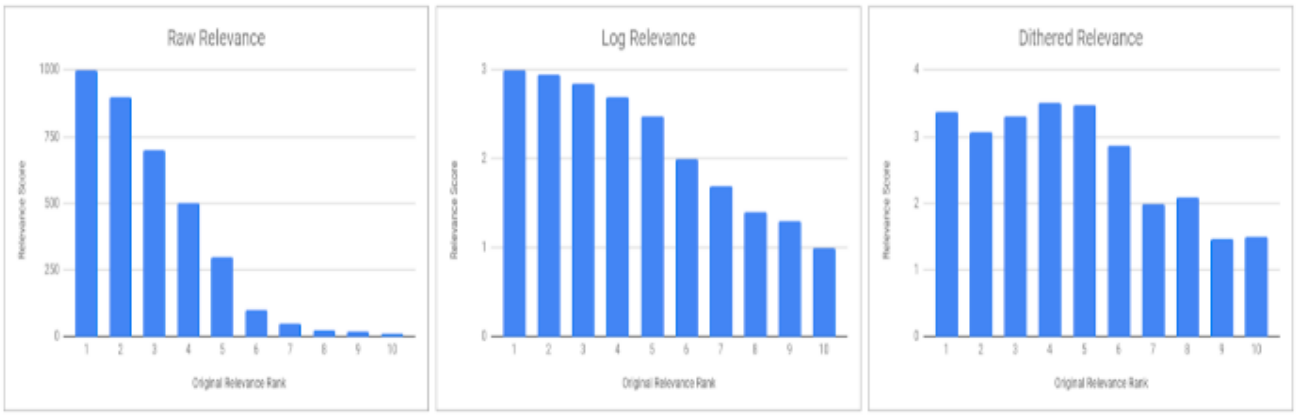
However, there are a few more challenges a good recommendation system has to overcome. Recommending the same things over and over is boring. Even worse, recommending the same things produces bad data and causes content fatigue.

Two simple and intuitive strategies to improve the value of recommendations are

- Anti-Flood: Penalise the second and third recommendations if they have the same similarity scores to the top recommendation.
- Dithering: Add a wildcard recommendation to create interesting new data points for the recommendation system to keep learning about other content.

These steps ensure an interesting user experience and new data on alternative

recommendations



REFRENCES

- https://towardsdatascience.com/how-to-build-a-recommendation-engine-quick-and-simple-aec8c71a823e
- https://towardsdatascience.com/learning-how-recommendation-system-recommends-45ad8a941a5a
- https://developers.google.com/machine-learning/recommendation
- Google training module on Recommendation Systems.

THANK YOU

shourya.mehrotra@gmail.com