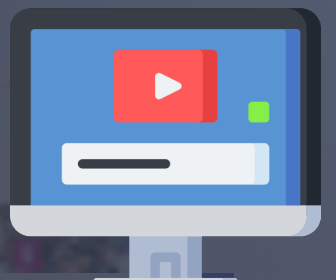
MINLP - Video Recommender

Agnish Upadhyay 200107005

Rishabh Lal 200107066



Introduction

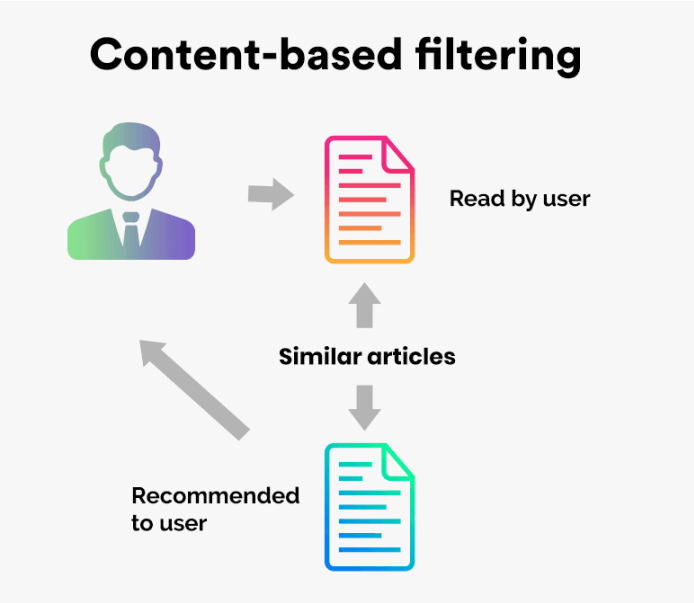
- Videos have gained popularity as a form of digital entertainment due to their effectiveness in storytelling and engagement. With the pandemic, video content has become a daily staple. However, the sheer amount of available content can make it difficult for viewers to find what they want, leading to disengagement.

- Thus, there is a need for algorithms that tailor suitable content. Video Recommendation Algorithms (VRAs) utilize data analysis techniques to analyze user behavior and preferences, providing personalized content recommendations. VRAs benefit not only viewers but also content providers and platforms by improving engagement and retention. By suggesting relevant content, VRAs prevent viewers from becoming disengaged or overwhelmed, leading to increased loyalty and revenue.

- We aim to model such an algorithm mathematically and display the most relevant results. These results will be calculated based on the similarities and the performance of a video, corresponding parameters to which have been given in our dataset.

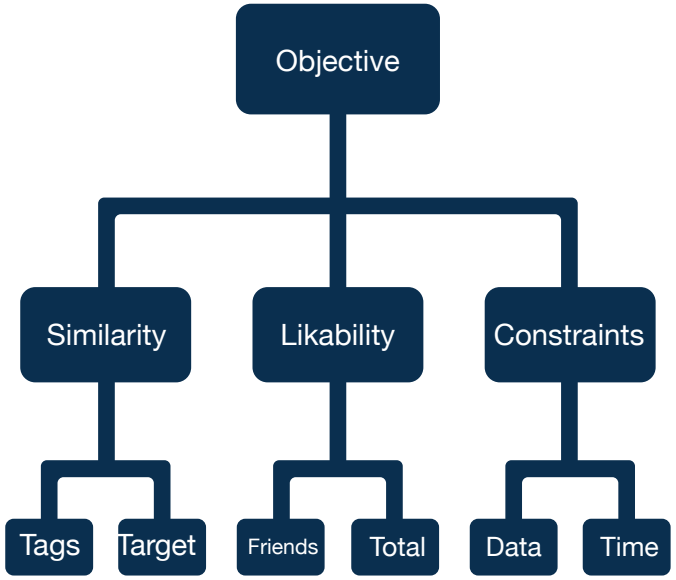
Problem Statement

The problem poses us to build a video recommendation system that uses video tags to suggest a set of 'n' videos to users based on their first clicked or target video. The system takes into account two main factors to determine user interest in each video: how similar it is to the target video and its overall likability. By analyzing these parameters, the recommendation system must identify the most intriguing videos to suggest. The solution must also adhere to time and internet constraints to be considered valid.



Objective

The problem is to be solved mathematically and then display the indices of the videos that are to be suggested. The resolution quality for each kind of video to also be shown. The parameters generated in the dataset should yield result even when changed. The most optimal solution must be calculated keeping in mind the constraints of the problem. It is essential that the algorithm runs fast enough such that it can fetch videos in real-time.



Mathematical Formulation

01

Quality Selection

The first equation tells the model which quality video must be chosen. Either one of the low, medium or high quality video has to be chosen for a particular index of the video.

We can see that the the sum equates to Z. Since Z is binary, either two of the three are 0 or all three are 0, indicating no choosing of the image itself.

$$Low_j + Med_j + High_j = Z_j$$

02

Friend Liking Indicator

This equations tells that F(j) should be 1 if friends like is greater than 0, and if friends like is 0, then F(j) = 0. For this purpose, signum function is used in the code implementation.

We needed an indicator to know whether a particular video has been liked by friends or not. This is important since it directly affects our objective function.

$$F_j = \begin{cases} 1, & \text{if } Like_{fr,j} \neq 0 \\ 0, & \text{if } Like_{fr,j} = 0 \end{cases}$$

Mathematical Formulation

03
Data Constraint

This equation tells that if video is available in particular quality, then its file size must be added up. The binary variables(i.e., Low(j), Med(j), High(j) are multiplied with the video quality and the sum should be less than the internet quota available to us.

$$\sum_{j=1}^{N} Quality_{i,\,j} * Type_j) \leq InternetUsage$$

where i=l,m,h

For i=l , $Type_j = Low_j$
For i=m , $Type_j = Med_j$
For i=h , $Type_j = High_j$

04
Total Likes

This creates an array of Total likes which is simply the sum of friend like and others like.This array will be used in our objective function.

$$TotalLikes_j = Like_{fr,\,j} + Like_{ot,\,j}$$

05
Display Constraint

This equation tells that the sum of all the videos that is shown to the user is equal to 'n',which is the number videos that we want to show. Since Z is binary, forcing sum of 5o binaries to 20 forms exactly 20 of them to be 1.

$$\sum_{j=1}^{N} Z_j = n$$

Mathematical Formulation

06

Anti-saturation provision

It tells the model that the number of medium quality videos must be equal to half the total videos that is shown to user.And accordingly, number of low and high quality videos are adjusted.This constraint is added in order to keep the user interested in watching the videos,because more no. of high quality videos might exhaust the internet available to us.And similarly,more number of low quality videos will cause the user to discontinue watching the available videos.

$$\sum_{j=1}^{N} med_j = n/2$$

07

Watchtime Constraint

This tells the model that the sum of the watchtime of all videos must be less than the recommended screentime, otherwise the user might get addicted to the social media platform.

Since the durations are only to be added when the video is actually suggested, Z is multiplied.

$$\sum_{j=1}^{N} Z_j * Watchtime_j \leq ScreenTime$$

Mathematical Formulation

08*

Auxiliary Equation

This constraint tells the model that the suggestion array(which contains the index of videos which is to be shown) should be equal to Z(j). Here Z(j) might be 0 or 1. '1' indicates that the video is suggested to be shown and '0' means that the video is not suggested. Used for displaying the final results.

$$Suggestion_j = Z_j$$

09

Choosing the server

This constraint tells that a particular video may be available in more than one servers and at most in all servers.The mathematical model must choose which server it uses to extract a particular video minimising the latency at the same time.

$$Z_j * (Server_{s1,j} + Server_{s2,j} + Server_{s3,j} \leq 3$$

10

Latency Constraint

It tells that if a video is extracted from a particular server than some latency time is spent.

Suppose, we have been given Average latency, then total no. of videos multiplied by the former should be greater than the latency time taken to extract a particular video from a particular server.

$$\sum_{j=1,d=1}^{N,3} Server_{d,j} * Latency_d * Z_j/1000) \leq AL * n$$

Objective Function

There are two dependencies of the objective function:

01

Similarity

Cosine similarity score is used for this part of the objective function. Cosine similarity is a metric used in Natural Language Processing to measure text-similarity between two documents, represented as n-dimensional vectors. The similarity score ranges from 0 to 1, with 1 indicating identical vectors and 0 indicating no similarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

In order to calculate the similarity score between videos, we similarly deployed the cosine similarity (which ranges between 0 and 1). Our approach involved calculating the similarity score of all other videos relative to a specific target video, with Video 1 serving as the current target. This similarity score was then integrated as a supplementary parameter in the GAMS model. If N is the number of videos in our dataset,

$$TotalSimilarityScore = 10 * \sum_{j=1}^{N} Similarity_j * Z_j$$

Objective Function

There are two dependencies of the objective function:

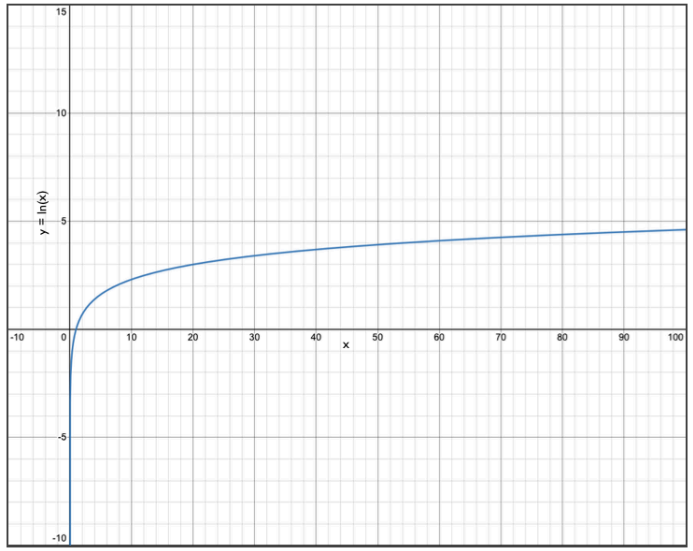
02

Likability

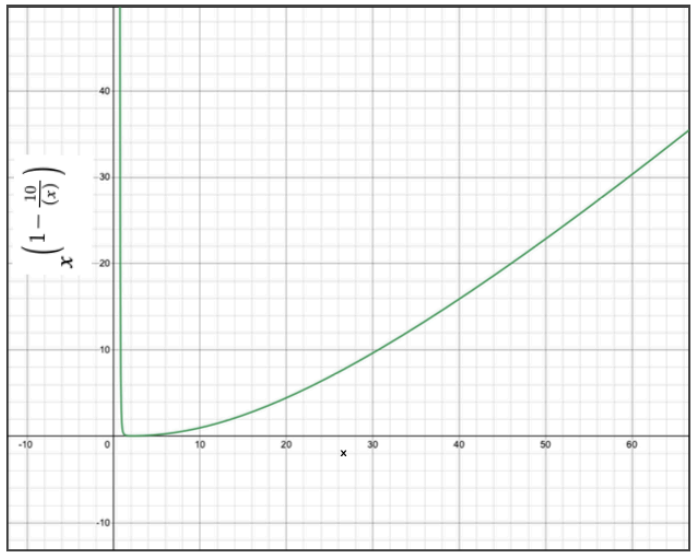
A video's performance for a new user depends on how well it has performed up till the present which is indicated by likes. Also, the relevance for a user is boosted if his/her friends or followers have already liked the video. So there are two dependencies for likability as well.

Total likes dependence: For highly-liked posts, quality may not increase much beyond a certain point due to factors like popularity bias and an abundance of content. Our model takes this into account by approximating the number of likes as constant once a threshold is reached. This behavior is represented by logarithmic function.

Friend likes dependence: Derived from Minkowski Distance and some heuristics, the function to the right represents dependence of likability on friend likes.



Total



Friends

Objective function

From the previous 2 slides, after weighing the two kinds of dependencies on the basis of the priorities:
Similarity Score > Friend Like Dependence > Total Like Dependence

We have formulated the objective function. Mathematically,

$$Obj = [\sum_{j=1}^{N} Similarity_j * Z_j + Likefactor]$$

$$Likability = \begin{cases} log(0.0001 + T/40) \\ 0.066 * L^{(1-10/(x+0.001))}, & \text{if } L \neq 0 \\ log(0.0001 + T/10), & \text{if } L = 0 \end{cases}$$

where Likefactor = $\sum_{j=1}^{N} Z_j * (1 - F_j * Likability_1 + F_j * (Likability_2 + Likability_3)$

Python Code for similarity score

```
In [1]: import numpy as np
        import pandas as pd
        import math
```

```
In [2]: df = pd.read_excel(r'C:\Users\acer\OPTIMISATION_PROJECT\Tags_content.xlsx',na_values=['\t'])
        print(df)
```

	id	type	number	tag1	tag2	tag3	tag4	tag5	tag6
0	0	V	4	t3	t6	t7	t9	NaN	NaN
1	1	V	4	t1	t5	t7	t10	NaN	NaN
2	2	V	5	t4	t6	t7	t8	t10	NaN
3	3	V	5	t1	t4	t8	t9	t10	NaN
4	4	V	3	t2	t6	t8	NaN	NaN	NaN
5	5	V	4	t1	t5	t7	t10	NaN	NaN
6	6	V	5	t3	t5	t8	t9	t10	NaN
7	7	V	3	t2	t4	t8	NaN	NaN	NaN
8	8	V	5	t4	t6	t7	t8	t10	NaN
9	9	V	4	t1	t8	t9	t10	NaN	NaN
10	10	V	3	t1	t2	t9	NaN	NaN	NaN
11	11	V	3	t4	t8	t9	NaN	NaN	NaN
12	12	V	2	t3	t9	NaN	NaN	NaN	NaN
13	13	V	5	t2	t4	t5	t6	t10	NaN
14	14	V	5	t2	t5	t7	t8	t9	NaN
15	15	V	5	t4	t5	t7	t9	t10	NaN
16	16	V	3	t6	t7	t9	NaN	NaN	NaN
17	17	V	3	t1	t6	t9	NaN	NaN	NaN
18	18	V	3	t4	t5	t9	NaN	NaN	NaN
19	19	V	3	t2	t4	t9	NaN	NaN	NaN
20	20	V	3	t4	t7	t10	NaN	NaN	NaN
21	21	V	2	t6	t9	NaN	NaN	NaN	NaN
22	22	V	3	t1	t5	t9	NaN	NaN	NaN
23	23	V	3	t3	t8	t9	NaN	NaN	NaN
24	24	V	4	t1	t5	t8	t9	NaN	NaN
25	25	V	4	t1	t4	t7	t10	NaN	NaN
26	26	V	2	t4	t9	NaN	NaN	NaN	NaN
27	27	V	4	t3	t8	t9	t10	NaN	NaN

Declarations

```
In [3]: #df['tags']=df['tag1']+','+df['tag2']+','+df['tag3']+','+df['tag4']+','+df['tag5']+','+df['tag6']+','+df['tag7']+','+df['tag8']+'
        df['tags']=df['tag1']
        df = df.fillna(value=0)
        #ar=[]

        for i in range(0,50):
            for j in range(4,9):
                if df.iat[i,j]!=0:
                    df.iat[i,9]=df.iat[i,9]+','+df.iat[i,j]


        print(df)
        #data_cell_1 = df.iat[0, 1]                    # Using .iat attribute
        #print(data_cell_1)
```

	id	type	number	tag1	tag2	tag3	tag4	tag5	tag6	tags
0	0	V	4	t3	t6	t7	t9	0	0	t3,t6,t7,t9
1	1	V	4	t1	t5	t7	t10	0	0	t1,t5,t7,t10
2	2	V	5	t4	t6	t7	t8	t10	0	t4,t6,t7,t8,t10
3	3	V	5	t1	t4	t8	t9	t10	0	t1,t4,t8,t9,t10
4	4	V	3	t2	t6	t8	0	0	0	t2,t6,t8
5	5	V	4	t1	t5	t7	t10	0	0	t1,t5,t7,t10
6	6	V	5	t3	t5	t8	t9	t10	0	t3,t5,t8,t9,t10
7	7	V	3	t2	t4	t8	0	0	0	t2,t4,t8
8	8	V	5	t4	t6	t7	t8	t10	0	t4,t6,t7,t8,t10
9	9	V	4	t1	t8	t9	t10	0	0	t1,t8,t9,t10
10	10	V	3	t1	t2	t9	0	0	0	t1,t2,t9
11	11	V	3	t4	t8	t9	0	0	0	t4,t8,t9
12	12	V	2	t3	t9	0	0	0	0	t3,t9
13	13	V	5	t2	t4	t5	t6	t10	0	t2,t4,t5,t6,t10
14	14	V	5	t2	t5	t7	t8	t9	0	t2,t5,t7,t8,t9
15	15	V	5	t4	t5	t7	t9	t10	0	t4,t5,t7,t9,t10
16	16	V	3	t6	t7	t9	0	0	0	t6,t7,t9
17	17	V	3	t1	t6	t9	0	0	0	t1,t6,t9
18	18	V	3	t4	t5	t9	0	0	0	t4,t5,t9

Python Code for similarity score

```
In [4]: df = df.drop(df.columns[[1,2,3,4,5,6,7,8]], axis=1)

In [5]: print(df)
```

	id	tags
0	0	t3,t6,t7,t9
1	1	t1,t5,t7,t10
2	2	t4,t6,t7,t8,t10
3	3	t1,t4,t8,t9,t10
4	4	t2,t6,t8
5	5	t1,t5,t7,t10
6	6	t3,t5,t8,t9,t10
7	7	t2,t4,t8
8	8	t4,t6,t7,t8,t10
9	9	t1,t8,t9,t10
10	10	t1,t2,t9
11	11	t4,t8,t9
12	12	t3,t9
13	13	t2,t4,t5,t6,t10
14	14	t2,t5,t7,t8,t9
15	15	t4,t5,t7,t9,t10
16	16	t6,t7,t9
17	17	t1,t6,t9
18	18	t4,t5,t9
19	19	t2,t4,t9
20	20	t4,t7,t10
21	21	t6,t9
22	22	t1,t5,t9
23	23	t3,t8,t9
24	24	t1,t5,t8,t9
25	25	t1,t4,t7,t10
26	26	t4,t9
27	27	t3,t8,t9,t10
28	28	t6,t7,t9
29	29	t4,t6,t7,t9
30	30	t5,t7,t8,t9

Declarations

Python Code for similarity score

```
In [6]: from sklearn.feature_extraction.text import CountVectorizer   # using CountVectorizer class
        cv = CountVectorizer(max_features=10,stop_words='english')  #taking out top 10 common words

In [7]: vectors = cv.fit_transform(df['tags']).toarray()

In [8]: print(vectors)
```

```
[[0 0 0 1 0 0 1 1 0 1]
 [1 1 0 0 0 1 0 1 0 0]
 [0 1 0 0 1 0 1 1 1 0]
 [1 1 0 0 1 0 0 0 1 1]
 [0 0 1 0 0 0 1 0 1 0]
 [1 1 0 0 0 1 0 1 0 0]
 [0 1 0 1 0 1 0 0 1 1]
 [0 0 1 0 1 0 0 0 1 0]
 [0 1 0 0 1 0 1 1 1 0]
 [1 1 0 0 0 0 0 0 1 1]
 [1 0 1 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 1 1]
 [0 0 0 1 0 0 0 0 0 1]
 [0 1 1 0 1 1 1 0 0 0]
 [0 0 1 0 0 1 0 1 1 1]
 [0 1 0 0 1 1 0 1 0 1]
 [0 0 0 0 0 0 1 1 0 1]
 [1 0 0 0 0 0 1 0 0 1]
 [0 0 0 0 1 1 0 0 0 1]
 [0 0 1 0 1 0 0 0 0 1]
 [0 1 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 1]
 [1 0 0 0 0 1 0 0 0 1]
 [0 0 0 1 0 0 0 0 1 1]
 [1 0 0 0 0 1 0 0 1 1]
 [1 1 0 0 1 0 0 1 0 0]
 [0 0 0 0 1 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 1 1]
 [0 0 0 0 0 0 1 1 0 1]
```

Vectorisation of tags

Python Code for similarity score

```
[0 0 0 1 0 0 0 0 0 1]
[1 1 0 0 0 0 1 0 0 1]
[0 1 0 0 0 0 1 0 1 0]
[0 0 0 0 1 1 0 0 1 0]
[0 0 0 1 0 1 1 0 0 0]
[0 0 0 0 0 1 0 1 0 0]
[0 0 1 0 0 0 0 1 0 1]
[0 1 1 0 1 0 0 1 1 1]
[1 0 1 0 0 0 0 0 0 1]
[0 0 0 1 0 0 0 1 1 0]
[0 1 0 0 0 1 1 1 0 0]
[0 1 0 0 1 0 1 1 1 0]]
```

Similarity score with target as first element

```
In [9]: cv.get_feature_names()
```

```
C:\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
ature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_
  warnings.warn(msg, category=FutureWarning)
```

```
Out[9]: ['t1', 't10', 't2', 't3', 't4', 't5', 't6', 't7', 't8', 't9']
```

```
In [10]: from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity(vectors)
similarity[0]
```

```
Out[10]: array([1.        , 0.25      , 0.4472136 , 0.2236068 , 0.28867513,
       0.25      , 0.4472136 , 0.        , 0.4472136 , 0.25      ,
       0.28867513, 0.28867513, 0.70710678, 0.2236068 , 0.4472136 ,
       0.4472136 , 0.8660254 , 0.57735027, 0.28867513, 0.28867513,
       0.28867513, 0.70710678, 0.28867513, 0.57735027, 0.25      ,
       0.25      , 0.35355339, 0.5       , 0.8660254 , 0.75      ,
       0.5       , 0.        , 0.25      , 0.28867513, 0.5       ,
       0.4472136 , 0.5       , 0.67082039, 0.70710678, 0.5       ,
       0.28867513, 0.        , 0.57735027, 0.35355339, 0.57735027,
       0.40824829, 0.28867513, 0.57735027, 0.5       , 0.4472136 ])
```

GAMS MINLP code

Declarations & dataset

```
Sets

j videos/1*50/,k likes/fr,ot/, d servers/s1,s2,s3/, t tags/t1,t2,t3,t4,t5,t6,t7,t8,t9,t10/, q quality/l,m,h/,c capacity/C1,C2,C3/;
Alias(j, jmod);

Table Like(k,j) Likes of video j at likes k
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19      20      21      22      23      24      25
fr      10      3       2       6       8       13      2       4       20      12      0       25      21      22      11      9       3       16      29      2       19      27      10      0       29
ot      101     450     413     497     321     214     305     315     352     452     491     186     136     379     48      159     94      436     84      120     281     434     454     473     218

Table Server(d,j) Server for jth video
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19      20      21      22      23      24      25
s1      0       1       1       0       1       1       0       0       0       1       0       1       1       1       0       0       1       1       1       0       0       1       0       0       0
s2      1       0       1       1       1       0       0       1       0       0       1       1       1       0       1       0       0       1       0       1       1       1       1       0       1
s3      1       1       1       0       1       0       1       1       1       0       1       1       1       0       1       1       1       1       0       0       1       1       1       1       1

Table Tags(t,j)  Tags for jth video
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19      20      21      22      23      24      25
t1      0       1       0       1       0       1       0       0       0       1       1       0       0       0       0       0       0       1       0       0       0       0       1       0       1
t2      0       0       0       0       1       0       0       1       0       0       1       0       0       1       1       0       0       0       0       1       0       0       0       0       0
t3      1       0       0       0       0       0       1       0       0       0       0       0       1       0       0       0       0       0       0       0       0       0       0       1       0
t4      0       0       1       1       0       0       0       1       1       0       0       1       0       1       0       1       0       0       1       1       1       0       0       0       0
t5      0       1       0       0       0       1       1       0       0       0       0       0       0       1       1       1       0       0       1       0       0       0       1       0       1
t6      1       0       1       0       1       0       0       0       1       0       0       0       0       1       0       0       1       1       0       0       0       1       0       0       0
t7      1       1       1       0       0       1       0       0       1       0       0       0       0       0       1       1       1       0       0       0       1       0       0       0       0
t8      0       0       1       1       1       0       1       1       1       1       0       1       0       0       1       0       0       0       0       0       0       0       0       1       1
t9      1       0       0       1       0       0       1       0       0       1       1       1       1       0       1       1       1       1       1       1       0       1       1       1       1
t10     0       1       1       1       0       1       1       0       1       1       0       0       0       1       0       1       0       0       0       0       1       0       0       0       0

Table Quality(q,j) File size for jth video
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17
l       3730    3413    2449    2399    3884    3018    4950    2513    4567    3749    2323    2536    4660    2127    4919    4494    2046
m       6955    6164    8597    8536    8947    6628    6616    7335    6618    8186    7196    8527    7602    8552    7838    8216    8307
h       10590   10501   12342   11063   11537   13807   11470   11440   10639   14715   12916   12990   14943   10890   14561   14475   12280

Parameters Latency(d) Each cache server latency time in milliseconds
/s1 250,s2 300,s3 500/;

Parameters Watchtime(j) of video j in seconds
/1   29, 2    14, 3    38, 4    38, 5   100, 6   106, 7    56, 8   112, 9    52, 10   68, 11   99, 12   79, 13   119, 14   68, 15   120, 16   90, 17   90, 18   113, 19  115, 20   54, 21   98, 22

Parameters Similarity(j) of video j between 0 to 1
/1  1,2 0.25,3  0.4472136,4 0.2236068 ,5 0.28867513,6  0.25,7  0.4472136 ,8  0,9 0.4472136 ,10  0.25,11  0.28867513,12  0.28867513,13  0.70710678,14  0.2236068 ,15  0.4472136 ,16  0.4472136 ,17  0.8660254

Scalar InternetUsage in megabytes
/1024000/;

Scalar target in index
/0/;

Scalar ScreenTime in seconds
/90000/;

Scalar n DisplaySize in numbers ...it must be even
/14/;

Scalar AL Average Latency in milliseconds
/530/;
```

Please note that the data set is not completely visible due to space restrictions.

GAMS MINLP code

Variables and equations

```
59
60  Variables Obj,Low(j),Med(j),High(j),F(j),Suggestion(j);
61  Positive Variables TotalLikes(j);
62  Binary Variables Low(j),Med(j),High(j),F(j),Z(j);
63
64  Equations
65  Quality_Eqn(j),Likability_Eqn(j),Time_constraint,Bandwidth_constraint,Eqn_TotalLikes(j),Eqn_ImageCount,Eqn2(j),Sol,Extra(j),Eqn_MedQualityCount,Eqn8;
66
67
68  Sol..Obj=E=10*sum(j,Similarity(j)*Z(j)) + sum(j,Z(j)*(1-F(j))*log(0.0001 + TotalLikes(j)/40) + F(j)*(log(0.0001 + TotalLikes(j)/10)) + F(j)*0.066*cvPower(Like('fr',j), 1-10/(0.001+Like('fr',j))));
69
70
71  Quality_Eqn(j)..Low(j)+Med(j)+High(j)=E=Z(j);
72
73
74  Likability_Eqn(j).. F(j) =e= sign(Like('fr',j));
75
76
77  Bandwidth_constraint..sum(j,Quality('l',j)*Low(j))+sum(j,Quality('m',j)*Med(j))+sum(j,Quality('h',j)*High(j))=L=InternetUsage;
78
79
80  Eqn_TotalLikes(j)..TotalLikes(j)=E=Like('fr',j)+Like('ot',j);
81
82
83  Eqn_ImageCount..sum(j,Z(j))=e=n;
84
85
86  Eqn_MedQualityCount..sum(j,med(j))=e=n/2;
87
88
89  Time_constraint..sum(j, Z(j)*Watchtime(j))=l= ScreenTime;
90
91
92  Extra(j)..Suggestion(j) =e= Z(j);
93
94
95  Eqn2(j)..Z(j)*(Server('s1',j)+Server('s2',j)+Server('s3',j))=L=3;
96
97
98  Eqn8..sum((d,j),Server(d,j)*Latency(d)*Z(j)/1000)=L=AL*n;
99
```

GAMS MINLP code

Display parameters

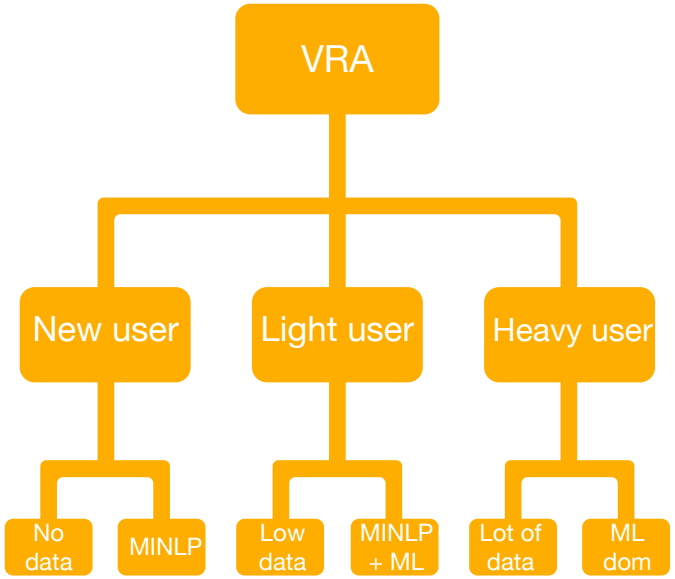
```
106
107  Model YouTube /all/;
108  Solve Youtube using MINLP maximizing Obj;
109  Display Suggestion.L,high.L,med.L,low.L;
```

Output

```
----    110 VARIABLE Suggestion.L   Indices to be shown

1  1.000,    13 1.000,    17 1.000,    18 1.000,    22 1.000,    24 1.000,    29 1.000,    30 1.000,    38 1.000,    39 1.000


----    110 VARIABLE Low.L          Low quality videos

13 1.000,    22 1.000


----    110 VARIABLE Med.L          Medium quality videos

1  1.000,    29 1.000,    30 1.000,    38 1.000,    39 1.000


----    110 VARIABLE High.L         High quality videos

17 1.000,    18 1.000,    24 1.000


EXECUTION TIME       =        0.404 SECONDS      4 MB  43.1.0 203303bb DEX-DEG
```

Conclusion

The GAMS implementation of the mathematical model showed promising results on the dataset, with scalability and adjustable parameters ensuring accurate output for varying inputs. It can work independently or alongside machine learning models for improved recommendations. The hybrid model combining pre-existing data with mathematical objective function values delivers better overall suggestions, highlighting the potential of mathematical modeling in recommendation systems. This success in solving a mixed integer non-linear programming problem underscores the significance of GAMS and the potential for further advancements in the field.



Probable Improvements

There is always room for improvement in any project, and the same holds true for ours. We have identified several areas where our recommendation system can be enhanced. Firstly, we can make a huge change by implementing dynamic fetching of data from the backend Python program responsible for calculating similarity scores. This will allow us to fetch updated data in real-time and provide more accurate recommendations to the user.

Secondly, it is possible to modify the algorithm to more heavily prioritize finding similarities between the user's previous video choices after the first few clicks. This will ensure that our recommendations are better tailored to the user's interests and preferences, and provide a more personalized experience.

Finally, there remains the possibility of using bitmasks to improve the space and time complexity of finding the similarity score. By utilizing bitmasks, we can optimize our recommendation system's efficiency and further reduce the processing time required to deliver accurate recommendations. Overall, these improvements will enhance the user experience and make our recommendation system even more effective.

AGNISH UPADHYAY 200107005
RISHABH LAL 200107066