## 1. The Problem

We want to build a recommendation engine to suggest content to the users that they are more likely to engage with. The content needs to be suggested based on the content previously engaged with, and content similar users engage in. Along with that the user's environment (recent attitude) should be gauged, and suggestions should be incorporated accordingly.

Although a user might not manually follow another user, the system should automatically recognise the people who’s content a user enjoys seeing. Let's develop a technique to measure this and call it an “Implicit follow” feature. We shall discuss about the implicit follow feature in the next section (2.3)

One of plans mentioned in the AI-ML infrastructure document was, *“As we move further, we would want to consider 2nd and 3rd order profiles as well.*For eg, content profiles of people a user follows, content profiles of people engaging with the same poll as the user.” This feature is the same as the user based collaborative filtering. This model can be incorporated

## 2. Solution Discussions

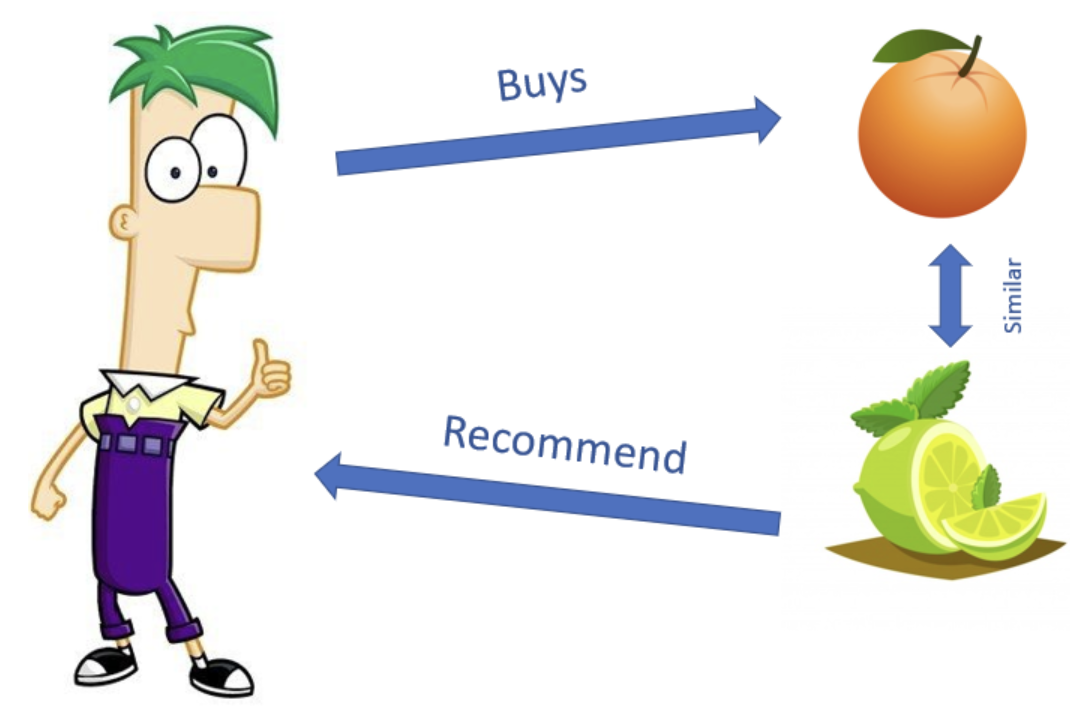
### 2.1 Hybrid Approach

To build a recommendation engine we will use a hybrid version of content based filtering & collaborative filtering.

#### 2.1.1 Content Based Filtering :

In the content-based recommendation system, we retrieve the poll's category and employ Natural Language Processing (NLP) to extract additional information such as the question's subject, tone, and other related features. By analyzing this feature set, we can discern a user's preferences for engaging with polls. For instance, if a user frequently engages with polls featuring whimsical questions about girlfriends, the attribute associated with "girlfriends" will be prominently reflected in the user's profile. This user profile, individually established for each user, serves as a representation of their preferences.

The recommendation process involves merging both the content and user profile, facilitating a more personalized and tailored recommendation.



Each user will have 3 different different groups of characteristics that the user likes. Each set of characteristic posts will ensure a certain type of engagement. Example :

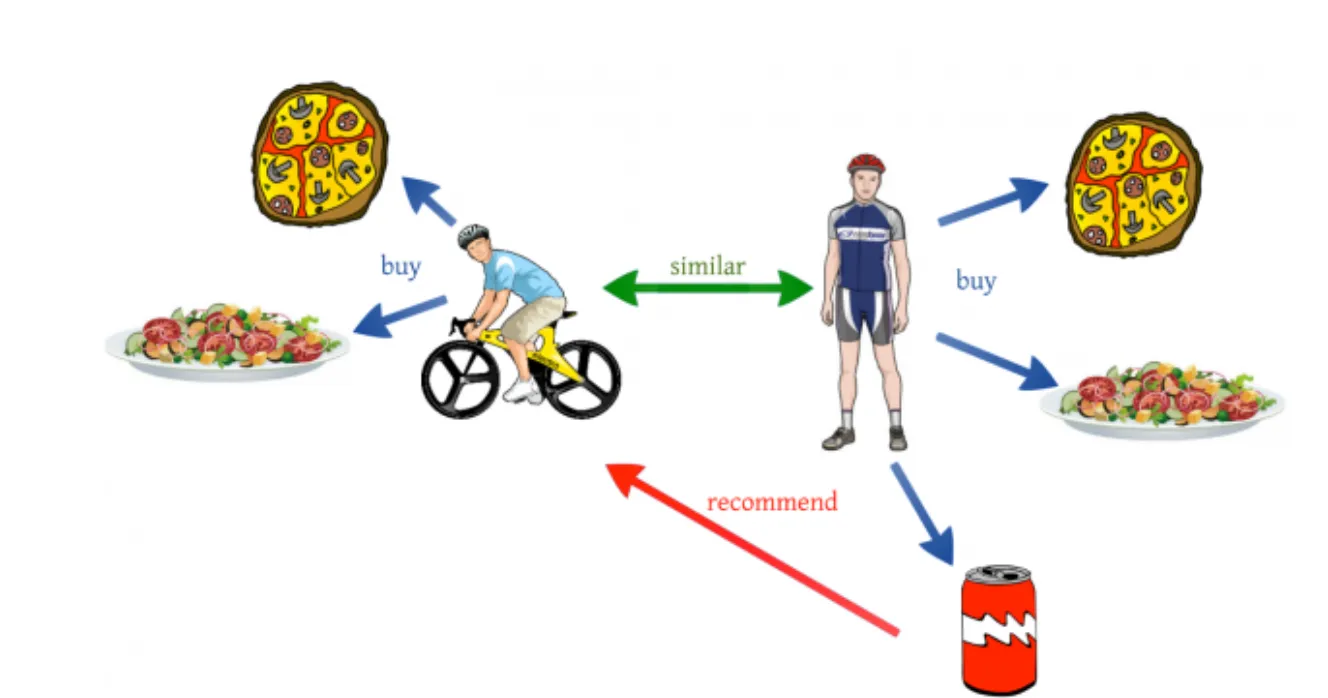
|  |  | **characteristic 1** | **characteristic 2** | **...** | **characteristic X** |
| --- | --- | --- | --- | --- | --- |
| *user1* | *overall* | w1\*(0.3) + w2\*(0.7) + w3\*(0.1) | –do– |  | –do– |
|  | *answered* | 0.3 | 0 | ... | 0.7 |
|  | *shares* | 0.7 | 0 | ... | 0 |
|  | *comment + like* | 0.1 | 0.5 | ... | 0 |

Where characteristic1 might be *subject\_girlfirend*, characteristic2 might be *subject\_Political*, and characteristic3 might be *would\_you\_rather\_tone*. Along with that we can have an overall engagement metric for a user, based on weights determined by the goals of the organization.

#### 2.1.2 User Based - Collaborative Filtering :

This profile will also include other features, such as the user's demographic and background information. At Level 1, user-user similarity can be measured using demographic information.

Following this, a user-item matrix will be created with users as rows and items as columns. This matrix will store the engagement of a particular user with all items. Similarity can be computed between the user and all other users to identify those with similar tastes. Therefore, if polls that the user has not yet engaged with have been engaged with by users similar to them, those polls can be assigned a higher propensity for engagement. Even for collaborative filtering we can product



### 

### 2.2 Session Attitude :

Depending on the user's behavior, setting, temperament, the engagement of the user with the polls will not be the same. For instance, during traveling (home-office) there is a high possibility that the user is only interested in reading comments written on polls. Thus, each session that the user has can be categorized differently. Some hypothesized ideas are shown below :

| **Attitude** | **Measure** | **Type of Recommendation** |
| --- | --- | --- |
| *Passive session* | Time spent on each poll can be used to determine the session type | Show posts that user expands and reads more often |
| *High active session* | Comments and Polls Answered | Show user posts with a high propensity to answer |
| *..* | .. | .. |
| *..* | .. | .. |
| *Frequent Interruption* | Multiple short sessions with high frequency | Show the user content with a high propensity to share  (User is engaged elsewhere too) |

The user's interaction in the first few minutes / first round of suggestion, can be used to categorize the users ‘Attitude’ using Machine Learning Techniques.

### 2.3 Implicit Follow :

Let's say there is a user A who has done a set of engagement in time ‘T’. And if user A’s engagements are similar to another user B’s engagements in a time period ‘∂T’ after ‘T’. Then we can say that B is implicitly following A. An algorithm can be made to find out implicit followers of an account. This data-point can be augmented with the features data point.

## 3. Model Selection and Approach for POC

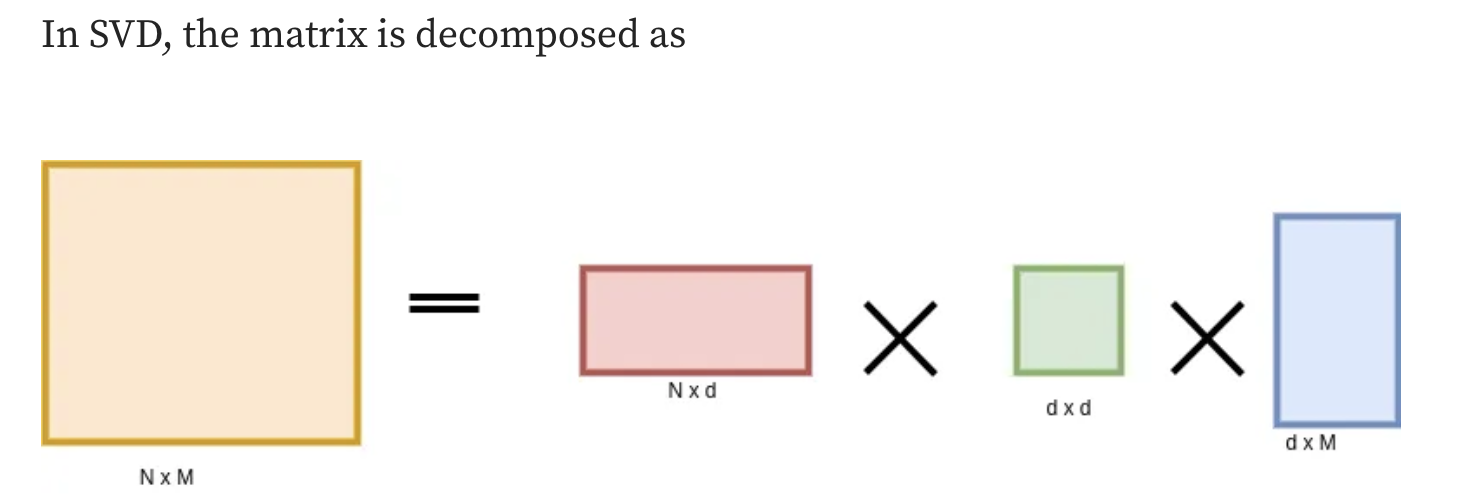
Based on literature, collaborative learning performs better than Content Based Learning. For a single model focused POC, the recommendation would be to select collaborative filtering.

#### Different Approach to Collaborative Filtering :

Other than directly finding similarity between users, based on from the user\_item engagement matrix, the following methods can be used :

* Clustering algorithms
* Matrix Factorization based algorithm
* Deep Learning methods

1. **Clustering Algorithms:** They normally use simple clustering Algorithms like K-Nearest Neighbours to find the K closest neighbors or embeddings given a user or an item embedding based on the similarity metrics used.
2. **Matrix Factorization based algorithms:** The user-item interaction matrix can also be factorized into two smaller matrices, and these two matrices can also be used to generate back the interaction matrix. So, we generate the factor matrices as feature matrices for users and items. These feature matrices serve as embeddings for each user and item. To create the feature matrices we need dimensional reduction.  
     
   The dimensionality reduction can be done by several methods:
   1. SVD: Singular Value Decomposition
   2. PMF: Probability Matrix Factorization
   3. NMF: Non-Negative Matrix Factorization



1. **Deep Learning Methods :**

SOTA (state of the art) Deep learning methods employ item features along with user\_item interaction. While providing weights to each item. Since all items do not classify users taste, it is able to put weights to those items. Elaborated below.

#### Why is deep learning better than Other approaches ?

Primary difference: Matrix factorization assumes user interactions (1 for positive feedback, 0 for no interaction) directly represent user preferences, even though this may not accurately reflect user likes or dislikes. In contrast, deep learning approaches consider the nuances of user engagement, acknowledging that an interaction (1) doesn't necessarily indicate liking the content, and 0 may simply mean non-engagement or missing data, highlighting the need to address the challenge of negative feedback.

#### 

#### Why to prefer Matrix Based Collaborative Filtering for POC ?

The approach suggested is to use the user\_item matrix to find user-user similarity. Rather than selection bias, there is an rejection bias, on why not to choose Deep Learning for POC -

* Data Scarcity: Deep learning models need substantial data, making them less suitable for scenarios with limited user-item interaction data.
* Interpretability: Deep learning models are often considered black boxes, lacking transparency in how recommendations are generated.
* Cold Start Problem: Deep learning models struggle with new users or items with limited interaction history, where traditional methods may perform better.
* Complexity: Neural networks can be resource-intensive requiring extended training time, and careful hyperparameter tuning.

Thus higher preference should be given to Matrix based collaborative learning approaches.

#### What are the primary and the secondary goals?

**Primary Goals :**

*Demonstrate Improved User Engagement:* Show that the recommendation engine enhances user engagement by delivering more relevant and personalized content.

Monitor - metrics such as increased user interactions, longer session durations, and higher content consumption. Pre-post analysis of user retention & increased frequency

Activities - Collect user feedback on the quality and relevance of recommendations. Evaluate if users find the recommended content valuable and engaging.

**Secondary Goals :**

*Validate Algorithm Suitability:* Assess the effectiveness of the chosen recommendation algorithms for the social media context.

Monitor - Experiment A/B test the POC approach based on the available data.

## 

## 4. Model Deployment

* Create Pipelines : Make pipeline for data to be preprocessed when called by an API. Expected the recommendation engine to update every time the user refreshes the app. Along with that set appropriate cool down, to prevent refresh until a certain time period.
* Containerization : The use of containerization tools like Docker to package the model and its dependencies, facilitating consistent deployment across different environments.
* Orchestration : Tools like Kubernetes for managing and automating the deployment, scaling, and operation of containerized applications.
* Monitoring and Logging: Systems for monitoring the performance of deployed models, logging metrics, and generating alerts for any anomalies or issues.
* Feedback Loop: Mechanisms to collect feedback from the deployed models to continuously improve and update them based on real-world performance and changing data patterns.

## 5. Maintenance | Feedback | Improvement

The table provides a comprehensive overview of the metrics relevant for assessing the performance and impact of a collaborative filtering recommendation engine in the context of social media post recommendations.

| **Business Perspective Metrics** | **Model Perspective Metrics** |
| --- | --- |
| 1. Click-Through Rate (CTR) | 1. Precision at K |
| 2. Likes, Shares, Comments | 2. Recall at K |
| 3. User Retention Rate | 3. F1 Score at K |
| 4. Conversion Rate | 4. Novelty Score |
| 5. Time Spent on Recommended Content | 5. Diversity Index |
|  | 6. Serendipity Score |
|  | 7. User Satisfaction Surveys |

The following formula can be used to measure the Novelty, Diversity & Serendipity Metrics of the Model.

## 

Along with model performance parameter, some key system performance parameters include -

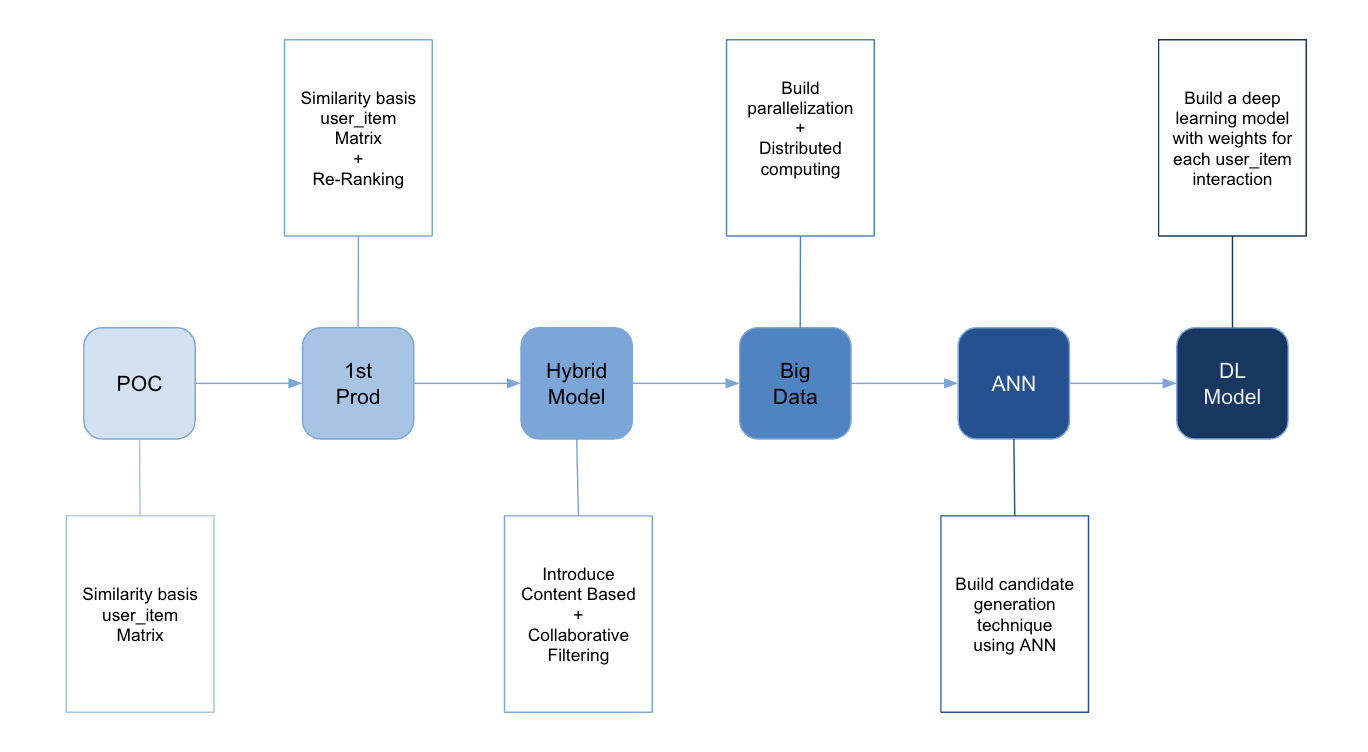
* Response Time: The response time of a recommendation system is the duration it takes to provide recommendations in response to a user query or request. This metric is critical as it directly influences the system's responsiveness, impacting the overall user experience by determining how quickly users receive relevant suggestions.
* Throughput: Throughput quantifies the system's capacity to generate recommendations within a specific timeframe. It reflects the system's ability to handle a substantial volume of user interactions efficiently. Higher throughput ensures that the recommendation engine can deliver recommendations promptly, even during periods of increased user activity.
* Resource Utilization: Resource utilization measures how efficiently the recommendation system utilizes computational resources, including CPU, memory, and storage, during the recommendation processes. Monitoring resource utilization is essential for optimizing performance, maintaining stability, and preventing resource-related bottlenecks.
* Latency: Latency refers to the delay or lag experienced by users when receiving recommendations. It directly impacts the real-time responsiveness of the system and influences user satisfaction. Lower latency contributes to a more seamless and instantaneous user experience.

Cohort analysis of these parameters can help us understand the next steps to be taken for model improvement and can act as a feedback loop.

## 

## 6. Big Data Issues when scaling

Scaling a recommendation engine for big data introduces challenges, including strain on computational resources, potentially impacting real-time responsiveness. Continuous monitoring of response time, throughput, and system latency is essential.



In handling millions of users and terabytes of data, scalability is crucial. The architecture must support horizontal scaling for both model training and recommendation inference. Techniques like Load Balancing are vital for effective computational workload distribution.

Parallel processing is key to efficient handling of extensive data volumes. Utilizing techniques like Data Parallelism, Model Parallelism, Task Parallelism, Pipeline Parallelism, GPU Acceleration, and Distributed Computing enhances efficiency, allowing concurrent processing.

Designing a recommendation system to seamlessly handle large-scale data involves strategic infrastructure implementation. Cloud-based solutions, distributed computing, and parallel processing algorithms are critical. Regular performance monitoring and optimization, including potential adjustments to the collaborative filtering algorithm, ensure optimal functionality as the system scales up.

#### 2.1.3 Approximate Nearest Neighbour :

Approximate Nearest Neighbors (ANN) techniques are employed in recommendation engines to efficiently find close neighbors or similar items for a given user or item in high-dimensional spaces. These methods, including locality-sensitive hashing (LSH) and tree-based structures like KD-trees or ball trees, provide a computationally efficient way to approximate nearest neighbors without exhaustively comparing all possible pairs. LSH, a popular ANN modeling technique, uses hash functions to map similar items to the same buckets, reducing the search space and accelerating the retrieval of approximate nearest neighbors in recommendation scenarios.

## 6. Big Data Issues when scaling

Scaling a Collaborative Filtering recommendation engine to handle big data introduces specific challenges they may encounter limitations that impact their effectiveness.

One notable challenge is the potential strain on computational resources. As the dataset expands, the collaborative filtering model may face difficulties efficiently processing and analyzing vast amounts of user-item interaction data. This could lead to longer processing times, potentially affecting the real-time responsiveness of the recommendation system. Constant monitoring of response time, throughput, resource utilization, and system latency.

In the context of millions of users generating terabytes of interaction data, the system's scalability becomes a crucial factor. The architecture must be designed to handle the increased load, ensuring that both model training and recommendation inference processes can scale horizontally to distribute the computational workload effectively. Load Balancing and similar Techniques must be applied

Parallel processing of user-item interaction data becomes essential. By leveraging parallel computing techniques, the recommendation engine can divide the dataset into smaller, manageable chunks and process them concurrently. This parallelization enhances the efficiency of both training the collaborative filtering model and generating real-time recommendations, addressing the challenges posed by extensive data volumes.

* Data Parallelism: Divide the dataset into smaller batches and process them independently.
* Model Parallelism: Split a large model into segments and process them concurrently.
* Task Parallelism: Break down the machine learning task into subtasks that can be executed concurrently.
* Pipeline Parallelism: Design a workflow with independent stages that operate concurrently.
* GPU Acceleration: Utilize Graphics Processing Units (GPUs) for parallelized matrix operations in deep learning.
* Distributed Computing: Distribute computation across multiple machines or nodes using frameworks like Apache Spark or cloud-based solutions.

Ensuring that the recommendation system can seamlessly handle millions of users and terabytes of interaction data involves strategic design and implementation of a scalable infrastructure. Cloud-based solutions, distributed computing frameworks, and efficient parallel processing algorithms are critical components for achieving a recommendation engine that can efficiently scale to meet the demands of large-scale collaborative filtering scenarios. Regular performance monitoring, optimization, and potential adjustments to the collaborative filtering algorithm may also be necessary to maintain optimal functionality as the recommendation system scales up to handle larger datasets and user bases.