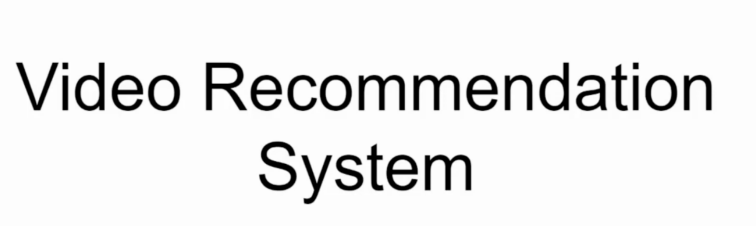
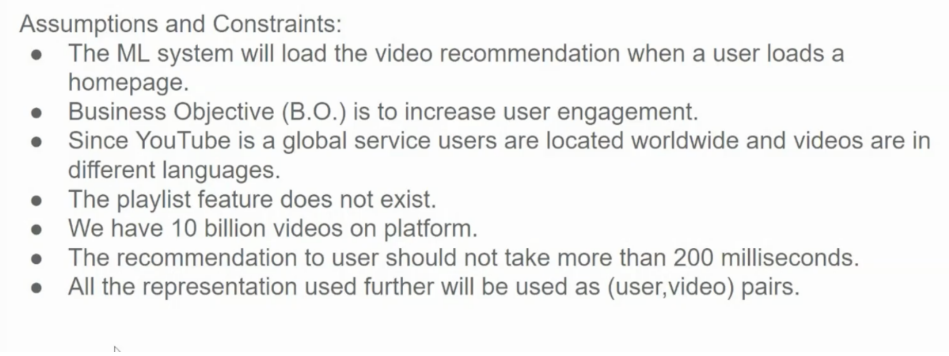
ML System Design Video Recommendation System

https://www.youtube.com/watch?v=Wo8ib5mBVKg&list=PL_b_MRp1BnEj4UuHv1jAGeO1a0VTRXsAk&index=10

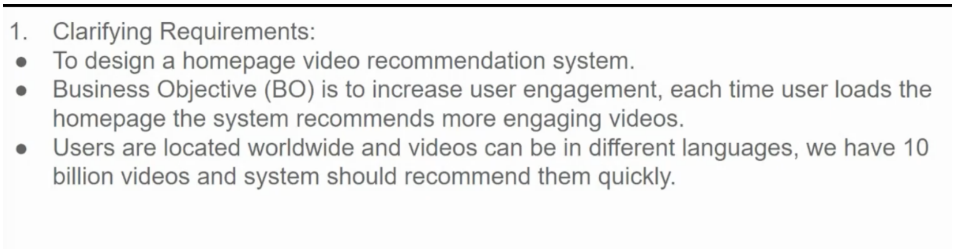
Objective



Assumptions and Constraints



Clarifying Requirements



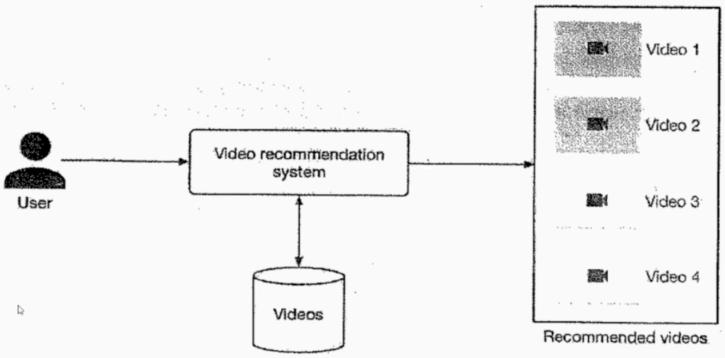
Framing as ML problem

2. Frame the Problem as an ML task:

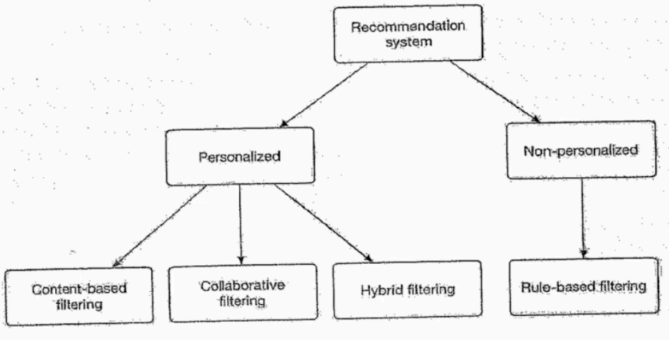
Defining the ML Objective: There are different approaches to translate BO into ML Obj. Some of them are:

- Maximize the number of clicks - the recommendation system is designed in a way to maximize user clicks.
  - The drawback being “clickbait” videos, which reduces user satisfaction and engagement overtime.
- Maximize number of completed videos - Recommending videos that the user is more likely to watch to completion.
  - But the recommendation system will start recommending videos of shorter duration which are quicker to watch.
- Maximize total watch time - recommends videos that user spend more time watching.
- Maximize the number of relevant videos - produces relevant videos for user but engineers or product managers have to define relevance.
  - This depends on user’s implicit and explicit reactions, eg. user explicitly pressing the like button or watches half of the video.
  - Once relevance is defined we can construct the dataset to train the model to predict a relevance score between a user and a video.
- We will use Maximize the number of relevant videos as we have more control over features to use and it does not have drawbacks of other options discussed earlier.

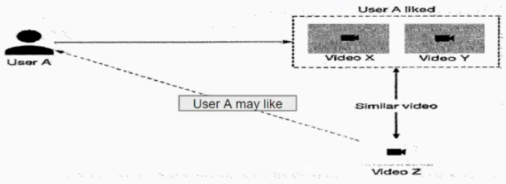
Specifying the System’s input and output:



3. Choosing the right ML Category:



1. Content-based filtering - This technique finds video to recommend new videos similar to those relevant in the past. E.g. If user has watched cooking videos then this technique will recommend more cooking videos.



Pros:

- Less dependency on making video profiles for user as video profile depends entirely on its features.
- Ability to capture unique interest of user as videos are recommended on user's previous engagement.

Cons:

- Difficult to capture user's new interest.
- Using this method requires domain knowledge as we often need to engineer features manually.

Understanding Content-Based Filtering

Content-Based Filtering is a recommendation technique that suggests items (in this case, videos) similar to those a user has previously interacted with. It's based on the intrinsic properties of the items themselves, rather than on user behavior.

How it works:

1. **Item Profile Creation:** Each video is assigned a profile based on its attributes like genre, director, actor, plot summary, keywords, or visual features extracted from the video frames.
2. **User Profile Creation:** A user profile is created based on the items the user has interacted with in the past. This profile can be as simple as a list of genres or keywords the user prefers.
3. **Recommendation Generation:** The system recommends items to the user based on the similarity between the user's profile and the item's profile. This similarity can be calculated using techniques like cosine similarity or Euclidean distance.

Pros of Content-Based Filtering:

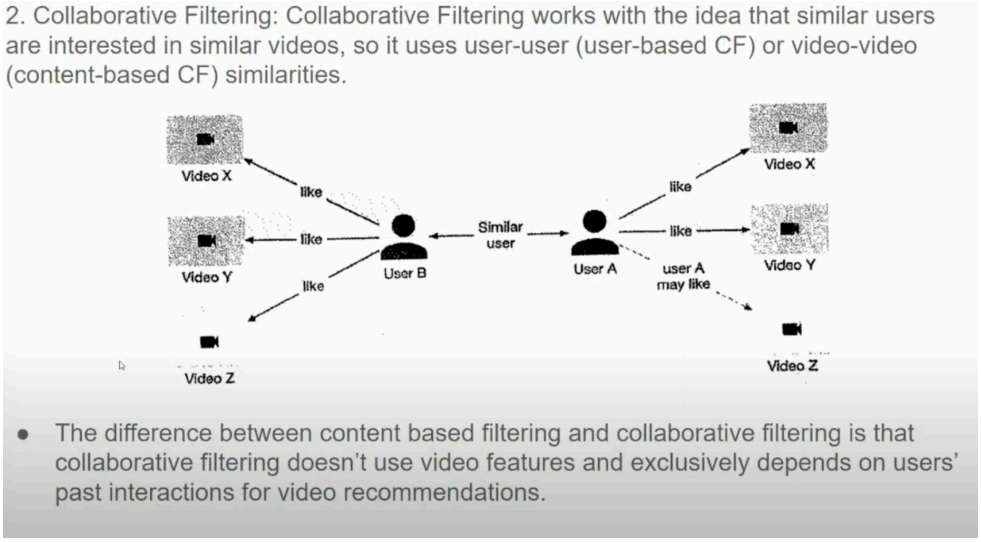
- **No Cold-Start Problem:** Can recommend items to new users without any historical data.
- **Transparent Recommendations:** Recommendations can be easily explained based on the item's attributes.
- **Serendipity:** Can introduce users to new items similar to their existing interests.

Cons of Content-Based Filtering:

- **Limited Novelty:** May recommend items that are too similar to what the user has already seen.
- **Requires Feature Engineering:** Manually defining relevant features can be time-consuming and requires domain expertise.
- **Overreliance on Item Metadata:** The quality of recommendations depends on the accuracy and completeness of item metadata.

In the context of the image:

- **User A** has watched **Video X** and **Video Y**.
- The system identifies similar videos based on their content, such as genre, director, or plot.
- **Video Z** is recommended to **User A** because it shares similar features with **Video X** and **Video Y**.



Collaborative Filtering is a recommendation system technique that leverages user-item interactions to make personalized recommendations. It works on the principle that users who have similar preferences in the past will likely have similar preferences in the future.

Types of Collaborative Filtering:

1. **User-Based Collaborative Filtering:**
    - **Similarity Measure:** Calculates the similarity between users based on their ratings or preferences for items.

    - **Recommendation:** Recommends items that similar users have liked.
2. **Item-Based Collaborative Filtering:**
    - **Similarity Measure:** Calculates the similarity between items based on how users have rated them.
    - **Recommendation:** Recommends items that are similar to items the user has liked in the past.

How it works in the image:

1. **User Similarity:** User A and User B have similar preferences, as they both like Video X and Video Y.
2. **Recommendation:** Since User A likes Video Z, the system recommends Video Z to User B, as they share similar preferences.

Advantages of Collaborative Filtering:

- **Personalization:** Can provide highly personalized recommendations.
- **No Need for Item Features:** Doesn't require explicit knowledge of item features.
- **Discovery of Serendipitous Items:** Can recommend items that users might not have considered otherwise.

Disadvantages of Collaborative Filtering:

- **Cold Start Problem:** Difficulty in recommending items to new users or for new items with limited user data.
- **Sparsity:** Real-world datasets are often sparse, which can impact the accuracy of similarity calculations.
- **Scalability:** Can be computationally expensive for large datasets.

Addressing the Limitations:

- **Hybrid Approaches:** Combining collaborative filtering with content-based filtering can address some of the limitations.
- **Contextual Bandits:** Using reinforcement learning to dynamically adjust recommendations based on user feedback.
- **Deep Learning Techniques:** Leveraging neural networks to capture complex patterns in user-item interactions.

Pros:

- Collaborative filtering does not rely on video features so it doesn't need domain knowledge to engineer feature from videos.
- Because collaborative filtering does not use video features they are faster and less compute intensive.
- Easy to discover users' new areas of interest as the system can recommend new videos about new topics that similar user's have engaged within the past.

Cons:

- Cold Start Problem - The system relies on users' past interaction to make new video recommendations but it lacks data for new users and suffers to recommend videos to new users. So when limited data is available the system cannot make accurate recommendations. This lack of user interaction data prevents collaborative filtering from finding similar users or videos.
- Cannot find niche interests - Collaborative filtering relies on finding similar users to make recommendations, it might be difficult to find similar users with specialized or niche interests.

Comparison for Content-based filtering and Collaborative filtering.

	Content-based filtering	Collaborative filtering
Handle New videos	✅	❌
Discover new videos	❌	✅
No Domain Knowledge necessary	❌	✅
Efficiency	❌	✅

The table provides a concise comparison of content-based filtering and collaborative filtering techniques, highlighting their strengths and weaknesses.

Let's break down the comparison:

Handle New Videos

- **Content-Based Filtering:** ✅
    - Can handle new videos effectively, as it relies on the video's content to make recommendations.
- **Collaborative Filtering:** ❌
    - Struggles with new videos, as it requires user interaction to build recommendations.

Discover New Videos

- **Content-Based Filtering:** ❌
    - May limit recommendations to similar items, potentially hindering discovery of new genres or interests.
- **Collaborative Filtering:** ✅
    - Can introduce users to new items based on the preferences of similar users.

No Domain Knowledge Necessary

- **Content-Based Filtering:** ❌
    - Requires domain knowledge to extract relevant features and similarities between items.
- **Collaborative Filtering:** ✅
    - Doesn't require explicit knowledge of item features, relying solely on user interactions.

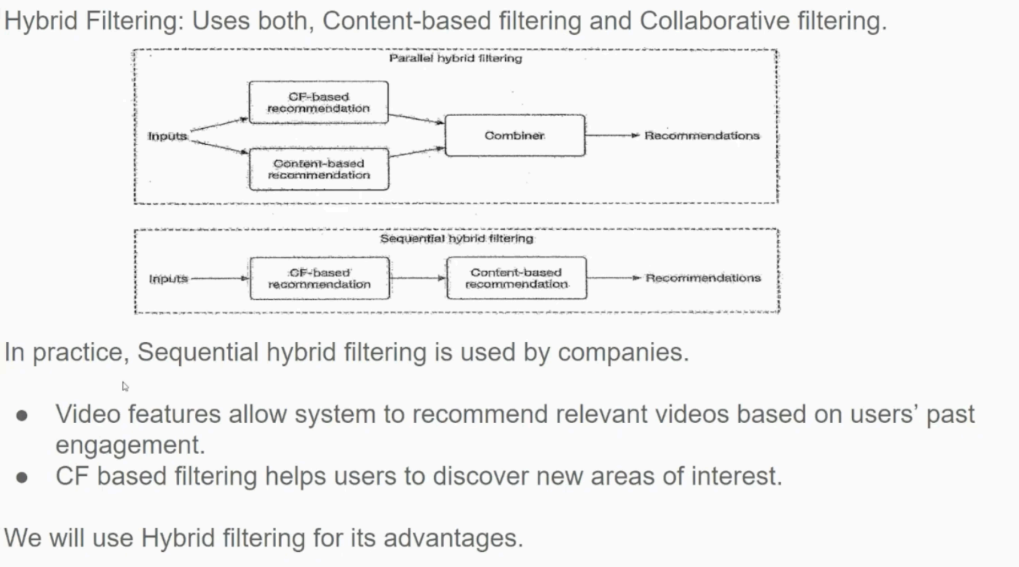
Efficiency

- **Content-Based Filtering:** ❌
    - Can be computationally expensive, especially for large datasets and complex feature extraction techniques.
- **Collaborative Filtering:** ✅
    - Generally more efficient, as it relies on simple similarity calculations between users or items.

In Summary:

- **Content-Based Filtering** is well-suited for scenarios where item metadata is rich and can be used to accurately represent the item's content. It's particularly effective for recommending similar items to users.
- **Collaborative Filtering** is better suited for scenarios where user behavior data is abundant and can be used to identify similar users and their preferences. It's particularly effective for discovering new items and personalizing recommendations.

Often, hybrid approaches that combine both techniques are used to overcome the limitations of each individual method and provide more robust and accurate recommendations.



Hybrid Filtering is a recommendation technique that combines the strengths of both Content-Based Filtering and Collaborative Filtering. By leveraging the benefits of both approaches, hybrid filtering can provide more accurate, diverse, and personalized recommendations.

Types of Hybrid Filtering:

1. **Parallel Hybrid Filtering:**
    - Both content-based and collaborative filtering techniques are applied independently to generate recommendations.
    - The results from both techniques are then combined, weighted, or voted upon to produce a final recommendation list.
2. **Sequential Hybrid Filtering:**
    - One technique is used to generate initial recommendations, and then the other technique is applied to refine the list.
    - For example, collaborative filtering can be used to generate a list of candidate items, and then content-based filtering can be used to rank these items based on their similarity to the user's preferences.

Advantages of Hybrid Filtering:

- **Combines strengths:** Leverages the strengths of both content-based and collaborative filtering.
- **Improved accuracy:** Can provide more accurate recommendations by considering both user preferences and item characteristics.
- **Reduced cold-start problem:** Can mitigate the cold-start problem by using content-based filtering for new users or items.
- **Increased diversity:** Can recommend a wider range of items, including both familiar and novel choices.

Real-world Application: Video Recommendation Systems

In the context of video recommendation systems, hybrid filtering can be used to:

- **Recommend similar videos:** Based on the content of the video (e.g., genre, director, actors).
- **Recommend videos watched by similar users:** Based on collaborative filtering.
- **Personalize recommendations:** Combine user preferences with item features to tailor recommendations to individual users.
- **Discover new interests:** Introduce users to new genres or content that they might not have considered otherwise.

Data Preparation:
Data Engineering:

- Video
- User
- User-Video interaction

1. Video:

Video ID	Length	Manual tags	Manual title	Likes	Views	Language
1	28	Dog, Family	Our lovely dog playing!	138	5300	English
2	300	Car, Oil	How to change your car oil?	5	250	Spanish
3	3600	Bali, Vlog	Our honeymoon to Bali	2200	255K	Arabic

2. User

ID	Username	Age	Gender	City	Country	Language	Time zone

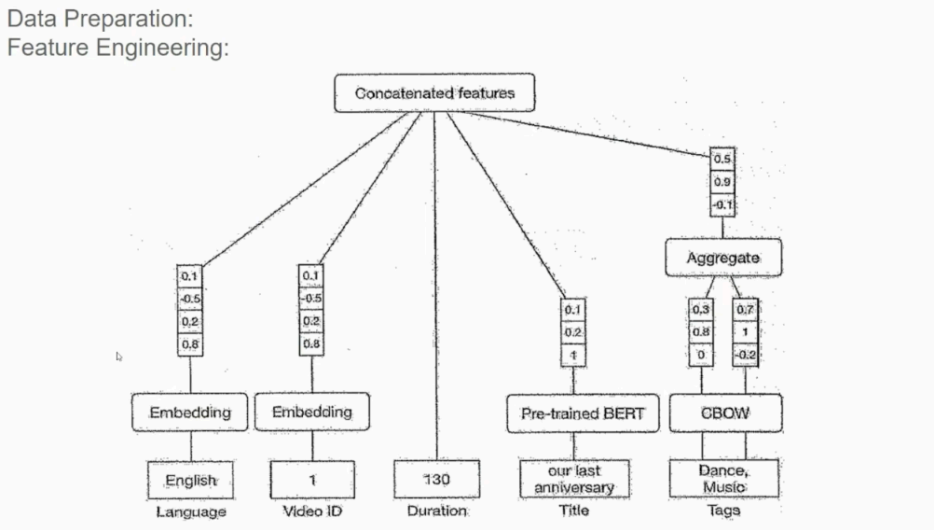
User-Video interaction:

User ID	Video ID	Interaction type	Interaction value	Location (lat, long)	Timestamp
4	18	Like	-	38.8951 -77.0364	1658451361
2	18	Impression	8 seconds	38.8951 -77.0364	1658451841
2	6	Watch	46 minutes	41.9241 -89.0389	1658822820
6	9	Click	-	22.7531 47.9642	1658832118
9	-	Search	Basics of clustering	22.7531 47.9642	1659259402
8	6	Comment	Amazing video. Thanks	37.5189 122.6405	1659244197

Data Preparation: Feature Engineering: 1. Video Feature, 2. User Feature.

1. Video Features:

- Video ID - IDs are categorical data and to represent them by numerical vectors, we use embedding layer.
- Duration - How long will the video last from start to finish. It is important information as some users prefer shorter videos while others prefer longer videos.
- Language - Language used in the video is an important feature as users naturally prefer particular languages. Language being a categorical variable and takes on a finite set of discrete values, we will use an embedding layer to represent it.
- Title and Tags - they are used to describe a video either added by user manually or implicitly added using a standalone ML model. They are valuable predictors as title of a video e.g. “How to paint a mountain”, “How to make pizza” implies the video is for drawing and cooking respectively.
    - For tags - we can use light weight pre-trained model of Continuous Bag of Words (CBOW).
    - For Title - we map it to feature vector to capture the contextual information of word embedding model such as BERT.



The diagram illustrates the process of preparing and engineering features for a machine learning model, likely for a video recommendation system.

Breakdown of the Steps:

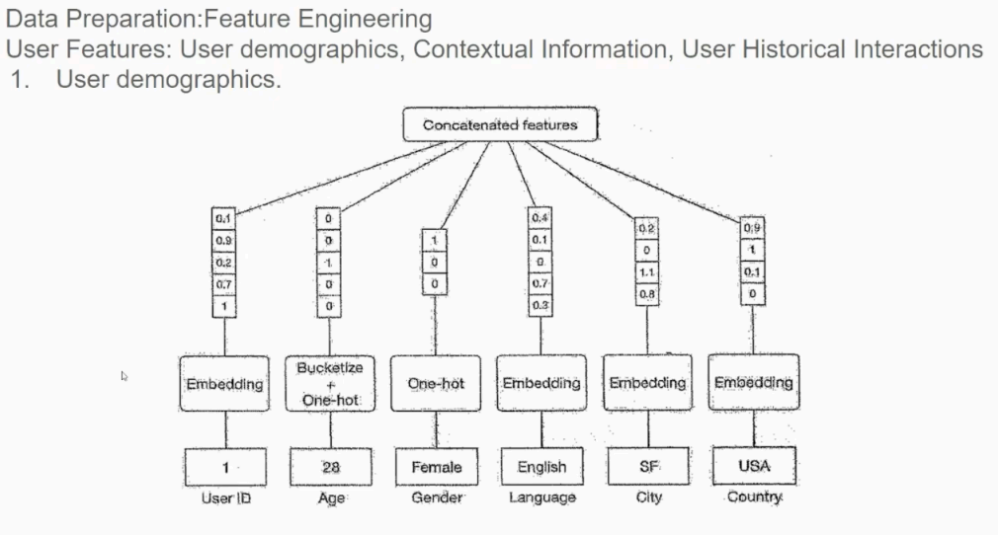
1. **Data Preparation:**
    - **Raw Data:** This includes various attributes of a video, such as language, video ID, duration, title, and tags.
2. **Feature Engineering:**
    - **Embedding:**
        - **Language:** The language of the video is converted into a numerical representation (embedding).
        - **Video ID:** A unique identifier for the video can be converted into a numerical representation.
        - **Duration:** The video duration can be converted into a numerical value.
    - **Pre-trained BERT:**
        - The video's title is processed using a pre-trained BERT model to extract semantic and syntactic information.
        - The output of the BERT model is a numerical representation of the title.
    - **CBOW:**

- The video's tags are processed using a Continuous Bag-of-Words (CBOW) model.
- The CBOW model learns the context of words based on their surrounding words. The output is a numerical representation of the tags.

- **Concatenated Features:**
  - The embeddings from different sources (language, video ID, duration, BERT, and CBOW) are concatenated into a single feature vector.
  - This combined feature vector represents the video in a rich and informative way.
- **Aggregate:**
  - The concatenated features are further processed using techniques like averaging, weighted averaging, or more complex aggregation methods to produce a final feature representation for the video.

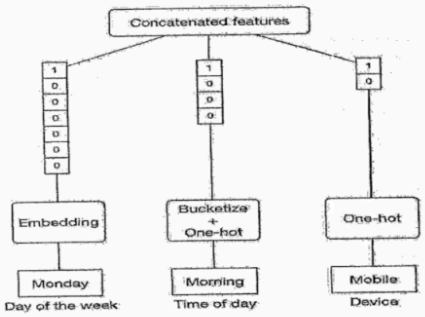
Purpose of Feature Engineering:

The goal of feature engineering is to transform raw data into meaningful features that can be used by a machine learning model. By combining different feature types (numerical, categorical, and textual), the model can learn complex patterns and relationships between videos.



2. Contextual Information:

- Time-of-day - A user may watch different videos at different time of day, e.g. working professional looking to improve themselves can watch more educational videos at evening.
- Devices - on mobile devices, users can prefer shorter videos compared to desktop users.
- Day of the week - depending on the day of the week, user can have different preferences for videos.

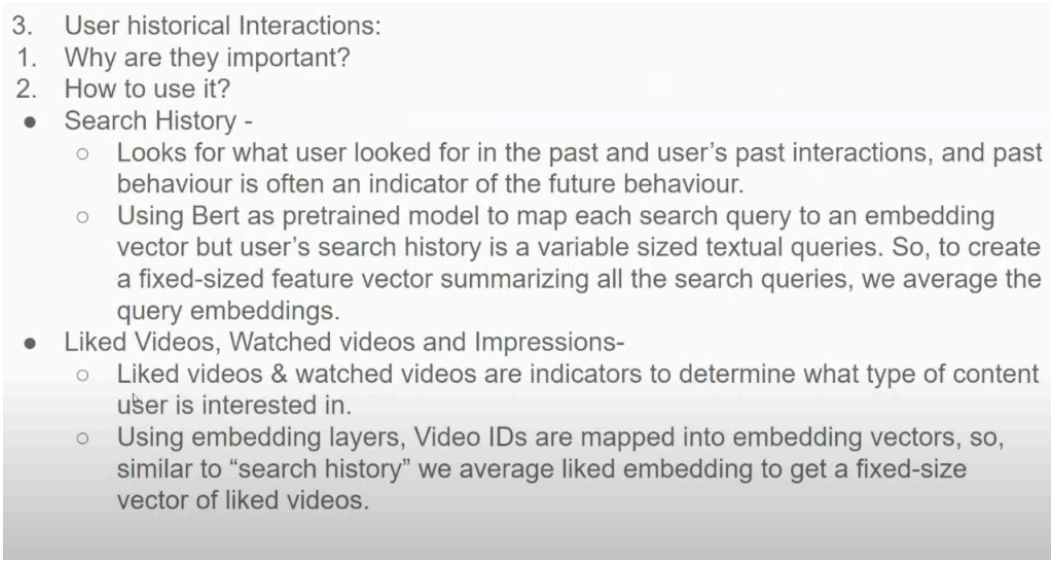


Breakdown of the Contextual Features:

1. **Time-of-Day:**
    - **Categorical Feature:** This feature can be categorized into time slots like morning, afternoon, evening, and night.
    - **Impact on Recommendations:** Different times of day can influence user preferences. For instance, during the day, users might prefer short, informative videos, while in the evening, they might opt for longer, entertainment-focused content.
2. **Device:**
    - **Categorical Feature:** This feature can be categorized into devices like mobile, tablet, and desktop.
    - **Impact on Recommendations:** Different devices have different screen sizes and user interactions. For example, shorter videos might be more suitable for mobile devices, while longer videos might be preferred on larger screens.
3. **Day of the Week:**
    - **Categorical Feature:** This feature can be categorized into weekdays and weekends.
    - **Impact on Recommendations:** User preferences might vary depending on the day of the week. For instance, users might prefer educational content on weekdays and entertainment content on weekends.

How to Incorporate Contextual Information:

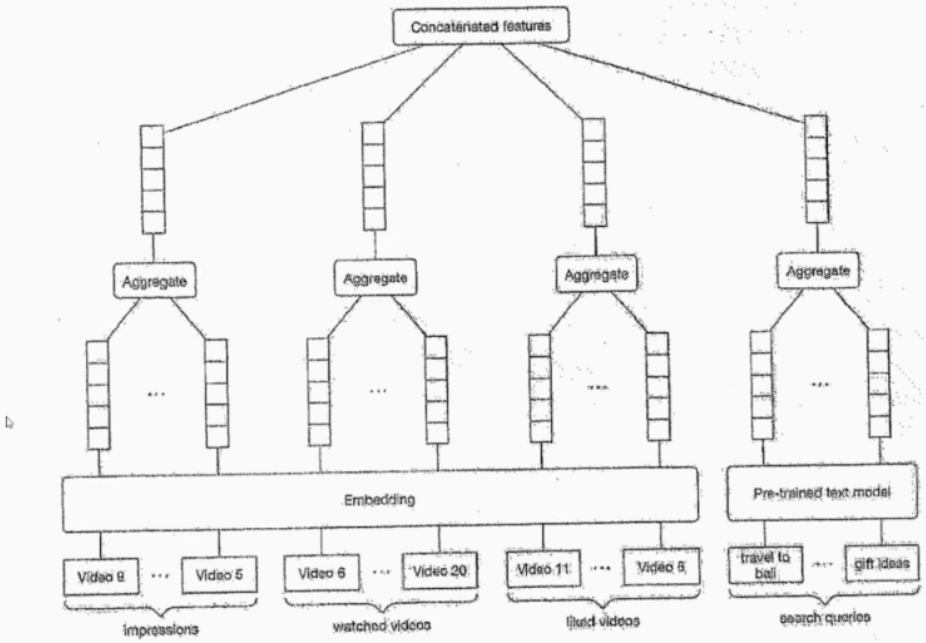
1. **Feature Engineering:**
    - **One-Hot Encoding:** Categorical features like day of the week, time of day, and device can be one-hot encoded to create binary features.
    - **Bucketing:** Numerical features like time can be bucketed into ranges (e.g., morning, afternoon, evening).
    - **Embedding:** Textual features like user preferences or video descriptions can be converted into dense vector representations using techniques like word embeddings or sentence embeddings.
2. **Model Training:**
    - **Contextual Features as Input:** These features can be used as additional input features to the recommendation model.
    - **Model Architecture:** The model can be designed to learn the impact of contextual features on user preferences and item relevance.


3. User historical Interactions:
1. Why are they important?
2. How to use it?

- Search History -
    - Looks for what user looked for in the past and user's past interactions, and past behaviour is often an indicator of the future behaviour.
    - Using Bert as pretrained model to map each search query to an embedding vector but user's search history is a variable sized textual queries. So, to create a fixed-sized feature vector summarizing all the search queries, we average the query embeddings.
- Liked Videos, Watched videos and Impressions-
    - Liked videos & watched videos are indicators to determine what type of content user is interested in.
    - Using embedding layers, Video IDs are mapped into embedding vectors, so, similar to "search history" we average liked embedding to get a fixed-size vector of liked videos.

Averaging the embeddings of 10 sentences creates a single vector that represents the **generalized semantic meaning** of those sentences. The resulting embedding captures the **common themes or overall context** across the sentences but loses the individual nuances or specific details of each one.

User-Video Interactions:



Model Development:

Common embedding based models for Content-based filtering or Collaborative based filtering:

1. Matrix Factorization
2. Two-Tower Neural Network

1. Matrix Factorization.
   a. Feedback Matrix.
   b. Matrix Factorization Models.
   c. Matrix Factorization Training.
   d. Matrix Factorization Optimization
   e. Matrix Factorization Inference

1. Matrix Factorization

a. Feedback Matrix:

- A matrix where rows represent users and columns represent items.
- Each cell contains a rating or preference value indicating the user's interaction with the item.

b. Matrix Factorization Models:

- Decomposes the feedback matrix into two lower-rank matrices:
    - **User Matrix:** Represents latent factors or features of users.
    - **Item Matrix:** Represents latent factors or features of items.
- The dot product of a user's latent vector and an item's latent vector predicts the user's rating for that item.

c. Matrix Factorization Training:

- The model is trained to minimize the difference between predicted ratings and actual ratings in the feedback matrix.
- Optimization techniques like gradient descent are used to update the model's parameters.

d. Matrix Factorization Optimization:

- Regularization techniques are often used to prevent overfitting and improve generalization.
- Techniques like Alternating Least Squares (ALS) or Stochastic Gradient Descent (SGD) can be used to optimize the model.

e. Matrix Factorization Inference:

- To make recommendations, the model calculates the dot product between the user's latent vector and the latent vectors of items.
- Items with the highest predicted ratings are recommended.

2. Two-Tower Neural Network

a. User and Item Towers:

- Separate neural networks are used to encode user and item information into dense embeddings.
- User embeddings capture user preferences and behavior.
- Item embeddings capture item features and attributes.

b. Similarity Calculation:

- The embeddings of the user and item are fed into a similarity function (e.g., cosine similarity or dot product) to calculate their similarity score.

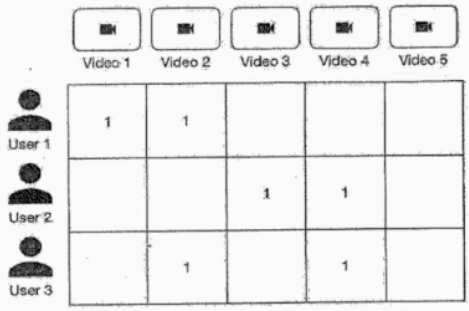
- Higher similarity scores indicate a better match between the user and the item.

c. Recommendation:

- Items with the highest similarity scores are recommended to the user.

Model Development:

1. Matrix Factorization : Feedback Matrix - Encases the user's opinion about a video.



Videos watched = 1 meaning "observed"or "positive".

A feedback matrix is built on user's opinion about video and interactions such as likes and shares.

Three ways to find how user finds recommended videos relevant:

- Explicit Feedback - tells us about user's opinion about the video that they have explicitly expressed. But has a major drawback as users don't always give explicit feedback making the matrix sparse and also making the ML model difficult to train.

Model Development:

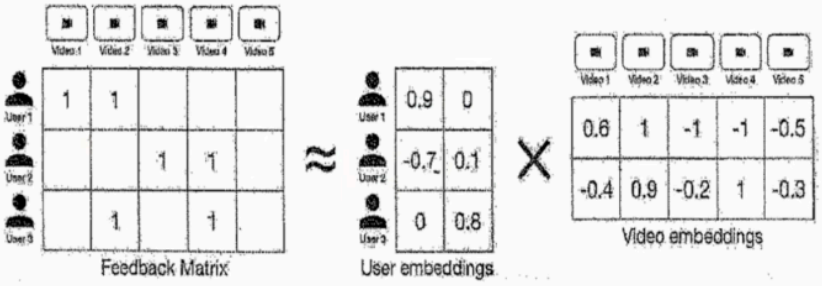
1. Matrix Factorization : Feedback Matrix - Encases the user's opinion about a video.

- Implicit Feedback - shows implicit indications of user's interest in the video like "click" or "watch time". With Implicit feedback more data points are available, training a better model. But cannot directly reflect user's opinion and similar to explicit feedback it can be noisy.
- Combination of Explicit and Implicit Feedback - combines the explicit and implicit feedback using heuristics.
    - Wiki's - **heuristic** (from Greek εὑρίσκω "I find, discover") is a technique designed for problem solving more quickly when classic methods are too slow for finding an exact or approximate solution.
- We will choose Combination of Explicit and Implicit as it aligns well with the ML objective to increase user engagement by giving more relevant videos and combination of both Explicit and Implicit is the best choice.

Model Development:

2. Matrix Factorization Model :

- It is a simple embedding model which decomposes the user-video feedback matrix into the product of two lower-dimensional matrices.
  - One lower dimensional matrix represents user embeddings mapping each user to embedding vector.
  - Other lower dimensional matrix represents video embeddings mapping each video to embedding vector
  - Both of their distance represents their relevance.



How it works:

1. **User-Item Rating Matrix:**
   - This matrix represents user-item interactions, where rows represent users, columns represent items, and the values in the cells represent ratings or preferences.
2. **Matrix Decomposition:**
   - The matrix is decomposed into two lower-dimensional matrices:
     - **User Matrix:** Represents latent features of users.
     - **Item Matrix:** Represents latent features of items.
3. **Prediction:**
   - To predict a user's rating for an item, we multiply the corresponding user and item latent vectors and sum their element-wise product.

Visualizing Matrix Factorization:

The provided image illustrates this process. The original user-item rating matrix is decomposed into two smaller matrices: a user matrix and an item matrix. The dot product of a user's latent vector and an item's latent vector gives a prediction of the user's rating for that item.

Advantages of Matrix Factorization:

- **Simple and Effective:** It's a relatively simple technique that can provide accurate recommendations.
- **Scalable:** Can handle large-scale datasets.
- **Cold-Start Problem:** Can be mitigated by incorporating additional information like item attributes or user demographics.

Limitations of Matrix Factorization:

- **Sparse Data:** It can struggle with sparse datasets, where many user-item ratings are missing.
- **Overfitting:** The model can overfit to the training data, especially if the latent dimensions are too large.

Addressing Limitations:

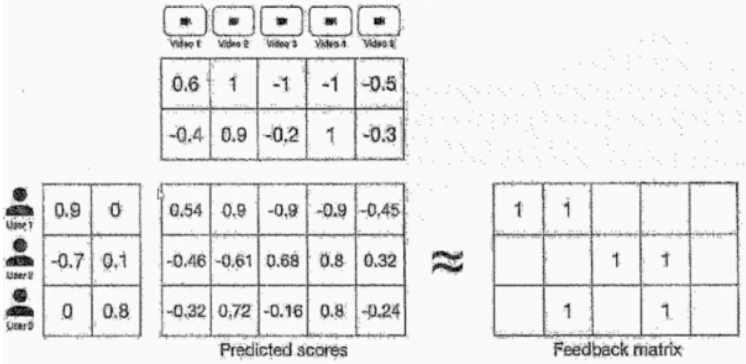
- **Regularization:** Adding regularization terms to the loss function can help prevent overfitting.
- **Feature Engineering:** Incorporating additional features like item attributes and user demographics can improve the model's performance.
- **Hybrid Approaches:** Combining matrix factorization with other techniques, such as content-based filtering or neural networks, can lead to more accurate recommendations.

Model Development:

3. Matrix Factorization Training : For training, we try to produce user and video embedding matrices which is good approximation to feedback matrix.

To learn this:

1. Matrix factorization first randomly initializes two embedding matrices.
2. Iteratively optimizes the embedding matrices to reduce the loss function between predicted scores and feedback matrix.



- Loss function selection is an important consideration here.

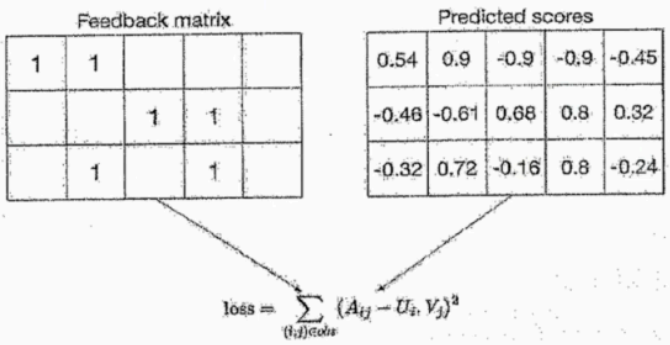
Model Development:

3. Matrix Factorization Training :

   Loss functions:

   a. Squared distance over observed (user,video) pairs.
   b. Squared distance over observed and unobserved (user,video) pairs.
   c. A weighted combination of Squared distance over observed and unobserved pairs.

a. Squared distance over observed (user,video) pairs.



- Measures the sum of squared distances over all pairs of observed (non-zero) entries in the feedback matrix.
- Only considers observed (non-zero) pairs which doesn't penalize the model for making bad predictions on unobserved pairs.
  - I.e for embeddings created for observed pairs of non-zero might work but same for unobserved pair might not work.

Matrix Factorization Training: Loss Functions

Matrix Factorization training aims to optimize the latent factors of users and items to minimize prediction error.

Common Loss Functions:

1. **Squared Distance Over Observed (User, Video) Pairs:**

   * Focuses on minimizing the error for observed ratings.

   * Formula:

   ```
   Loss = Σ(R_ui - <U_i, V_j>)^2
   ```

   * Where:

   - `R_ui`: Actual rating of user `i` for item `j`

- `<U_i, V_j>`: Predicted rating based on the dot product of user `i`'s and item `j`'s latent vectors

2. **Squared Distance Over Observed and Unobserved (User, Video) Pairs:**

* Minimizes error for both observed and unobserved pairs.

* Formula:

```
Loss = Σ(R_ui - <U_i, V_j>)^2 + λ Σ(<U_i, V_j>)^2
```

* The second term is a regularization term to prevent overfitting.

3. **Weighted Combination of Squared Distances:**

* Combines the previous two approaches.

* Formula:

```
Loss = α * Σ(R_ui - <U_i, V_j>)^2 + (1-α) * Σ(<U_i, V_j>)^2
```

* `α` is a weight parameter to balance the importance of observed and unobserved pairs.

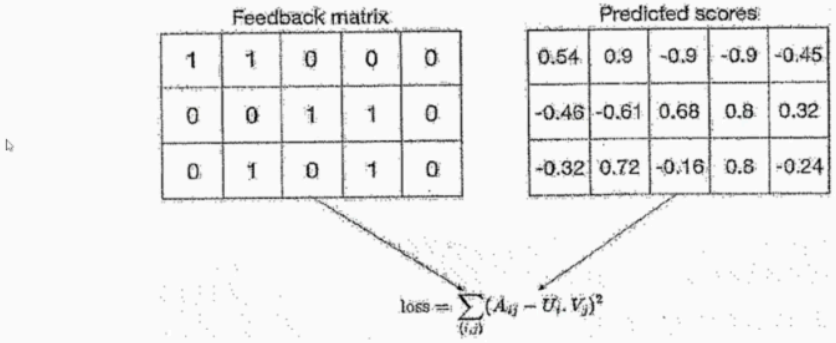
Choosing the Right Loss Function:

* Consider data sparsity, overfitting, computational cost, and desired outcome.

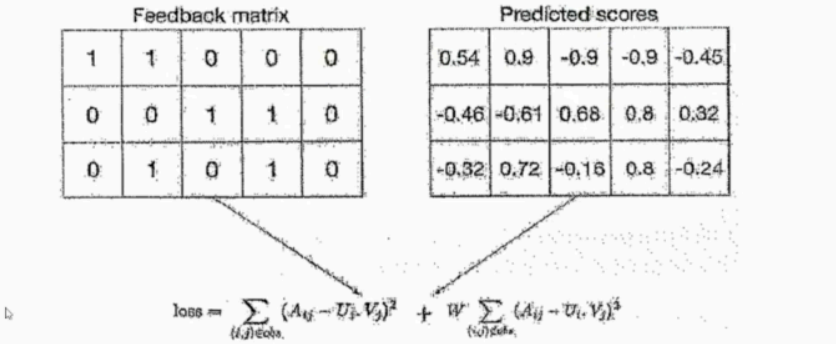
By carefully selecting and tuning the loss function, we can improve the performance of matrix factorization models and deliver more accurate and personalized recommendations.

b. Squared distance over observed and unobserved (user,video) pairs.

- This loss function assigns the negatives to unobserved pairs and assigns a zero in the feedback matrix which solves the issue of not penalizing model for bad predictions in previous loss function.
- Major drawback - there are lots of unobserved pairs making the feedback matrix sparse which results in unobserved pairs dominating the training, creating predictions mostly close to zero leading to poor generalization performance on unobserved (user,video) pair.



C. Weighted combination of Squared distance over observed and unobserved (user,video) pairs.



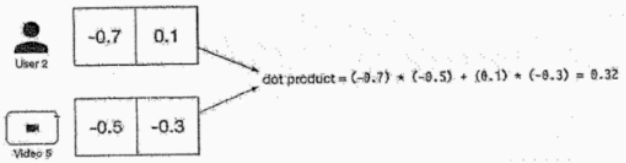
- First $\sum$ is for observed pairs and second $\sum$ is for unobserved pair with W, which is a hyperparameter that weighs two summations ensuring one does not dominates the other in training phase. A properly tuned W works best in practise.
- We will use this method.

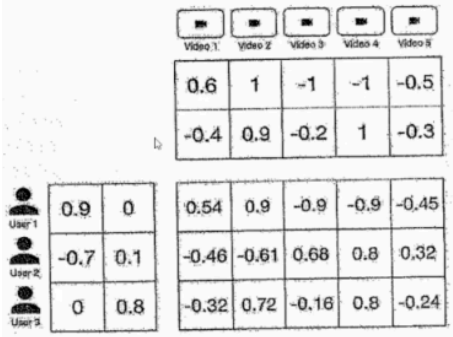
4. Matrix Factorization Optimization: for training an ML model, we require optimization algorithm.

- Stochastic Gradient Descent - used for minimizing the loss.
- Weighted Alternating Least Squares (WALS) - it is specific to matrix factorization.
  Order in which WALS work:
  - Fix one embedding matrix (U) and optimize other embedding (V).
  - Fix another embedding matrix (V) and optimize other embedding (U).
  - Repeat.
  - WALS converges faster and is parallelizable.
  - We will user WALS.

Model Development : Matrix Factorization

5. Matrix Inference: A dot product for similarity measure of random user embedding and a video embedding is calculated to find the relevance between them.





- System returns videos based on relevance scores.

Matrix Factorization uses user-video interaction only, so it is used in collaborative filtering.

Model Development: Matrix Factorization.
Pros:

- Faster training speed due having two matrices to learn.
- The serving time is also fast as the learned embeddings are static, meaning ones we learnt them, we can reuse them and do not need to transform the input at query time.

Cons:

- It relies on user-video interactions only and does not consider other important features as user's age, language or location which limits the predictive capability of the model and features like language can help us improve the quality of recommendations.
- Handling new user is difficult as there is not enough interactions for the model to generate meaningful embeddings. So, Matrix Factorization cannot determine whether video is relevant to a user by computing the dot product between their embeddings.

Pros of Matrix Factorization:

1. **Faster Training Speed:**
    - By decomposing the user-item rating matrix into two smaller matrices, matrix factorization can be trained efficiently.
    - This is especially beneficial for large-scale datasets.
2. **Efficient Inference:**
    - Once the model is trained, the learned embeddings can be stored and reused.
    - This allows for fast inference, as the model doesn't need to be re-trained for each query.

Cons of Matrix Factorization:

1. **Reliance on User-Item Interactions:**
    - The model's performance heavily relies on the availability of user-item ratings.
    - It struggles with cold-start problems, where there's limited data for new users or items.
2. **Limited Contextual Understanding:**

- Matrix factorization primarily considers user-item interactions and doesn't explicitly incorporate additional contextual information like user demographics, item attributes, or temporal factors.
- This can limit the model's ability to make accurate recommendations, especially for users with limited interaction history or for items with few ratings.

Addressing the Limitations:

To mitigate these limitations, hybrid approaches that combine matrix factorization with other techniques can be employed:

- **Hybrid Recommendation Systems:**
  - Incorporate content-based filtering to leverage item attributes.
  - Utilize contextual information like user demographics, time of day, and device type.
  - Employ collaborative filtering to leverage user-item interactions.
- **Deep Learning-Based Models:**
  - Neural network models can capture complex relationships between users and items.
  - They can handle large-scale datasets and incorporate various features, including textual, visual, and temporal information.

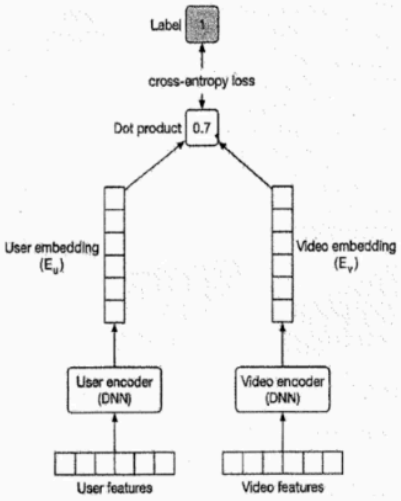
Model Development: Two-Tower Neural Network
Constructing the Dataset:

- We construct a dataset by labelling positive and negative from different (user,video) pairs based on the user's feedback.
  - Positive if user has explicitly liked the video or watched at least half of the video.
  - For Negative, we can randomly choose the videos that were not relevant or user explicitly disliked the video.

#	User-related features	Video-related features	Label
1	0 0 1 0.7 -0.6 0 0	0 1 0 0.9 0.9 1	1 (positive)
2	0 1 1 0.2 0.1 1 0	0 1 0 -0.1 0.3 1	0 (negative)

- While constructing the dataset user only finds small fraction of videos relevant making more negative pairs (majority) than positive pairs (minority) which constructs an imbalance dataset. We can treat an imbalance dataset in different ways.

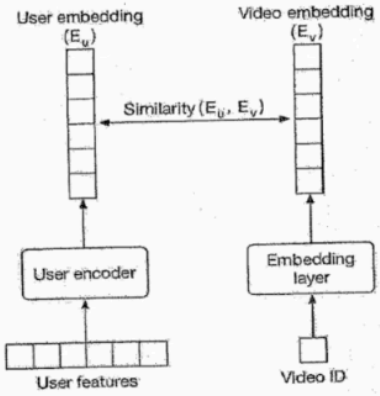
Model Development: Two-Tower Neural Network
Choosing the Loss function: Since two-tower neural network is trained to predict binary labels it can be categorized as classification task and use cross-entropy to optimize encoders during training.



Model Development: Two-Tower Neural Network

Two-Tower Neural Network Inference:

- At inference time, The video embeddings will be used to find the k most relevant videos making it a classic “k nearest neighbour” problem. We will use Approximate Nearest Neighbour (ANN) to find the most relevant videos efficiently.
- Two-Tower Neural Network is used in both Content-based filtering and Collaborative Filtering but when used in Collaborative filtering the video encoder is used as an embedding layer.



Two-Tower Neural Network Inference is the process of using a trained two-tower model to generate recommendations. As depicted in the image, it involves the following steps:

1. **Embedding Generation:**
    - **User Embedding:** The user's features (e.g., demographics, past behavior) are fed into the user tower to generate a dense vector representation, or embedding.
    - **Item Embedding:** The item's features (e.g., title, description, genre, tags) are fed into the item tower to generate an item embedding.
2. **Similarity Calculation:**
    - The user embedding and item embeddings are compared using a similarity metric, such as cosine similarity or dot product.
    - This similarity score indicates how relevant an item is to the user's preferences.
3. **Ranking and Recommendation:**
    - Items with the highest similarity scores are ranked and presented to the user as recommendations.

The two-tower model is a neural network architecture commonly used in recommendation systems, including content-based filtering. It consists of two separate neural networks, often referred to as "towers."

How it works in Content-Based Filtering:

1. **Item Tower:**
    - Takes as input the features of an item, such as genre, director, keywords, or visual features extracted from the video frames.
    - Processes these features through multiple layers of neural networks to generate a dense vector representation, or embedding, of the item.
2. **Query Tower:**
    - Takes as input the user's query or historical preferences (e.g., previously watched videos, genres of interest).
    - Processes this information to generate a user embedding.
3. **Similarity Calculation:**
    - The embeddings from both towers are compared using a similarity metric, such as cosine similarity or dot product.
    - Items with the highest similarity scores to the user's query embedding are recommended.

Advantages of Two-Tower Models for Content-Based Filtering:

- **Flexibility:** Can handle various types of features, including textual, numerical, and categorical.
- **Scalability:** Can be scaled to handle large datasets and complex models.
- **Personalization:** Can be adapted to personalized recommendations by incorporating user-specific information into the query tower.
- **Cold-Start Problem Mitigation:** Can be combined with collaborative filtering techniques to address the cold-start problem.

Example:

Consider a video recommendation system where we want to recommend videos based on their content. We can use a two-tower model as follows:

1. **Item Tower:**
    - Input: Video title, description, genre, director, and visual features.
    - Output: A dense vector representing the video's semantic and visual features.
2. **Query Tower:**
    - Input: User's search query or previously watched videos.
    - Output: A dense vector representing the user's intent or preferences.
3. **Similarity Calculation:**
    - Calculate the similarity between the query embedding and each item embedding.
    - Rank the items based on their similarity scores and recommend the top-ranked items.

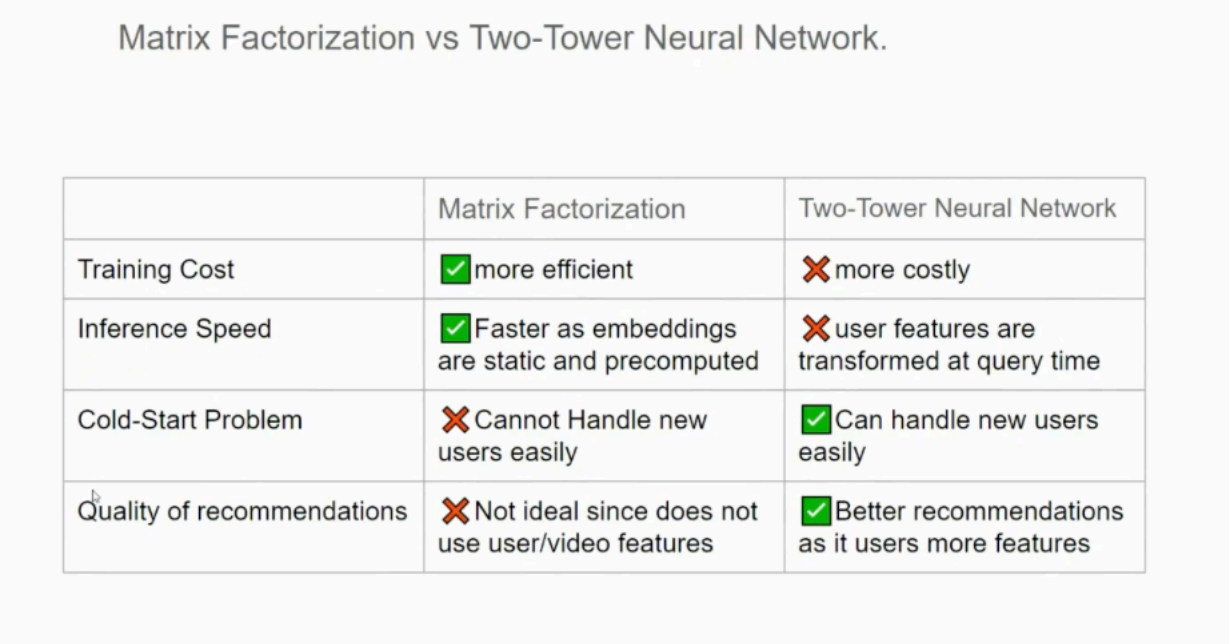
Model Development: Two-Tower Neural Network

Pros:

- The model accepts user feature such as age and gender which are used to make better recommendations.
- As user features are used the model handles new users easily.

Cons:

- The model needs to compute embeddings at query time making it slower. If we use content-based filtering, the model needs to transform video to video embeddings increasing the inference time.
- The training is more compute intensive as Two-tower Neural Network has more learning parameters than Matrix Factorization



The table provides a clear comparison of the strengths and weaknesses of Matrix Factorization and Two-Tower Neural Network models for recommendation systems. Let's break down the key points:

Training Cost

- **Matrix Factorization:** More efficient as it involves training a relatively simpler model with fewer parameters.
- **Two-Tower Neural Network:** More costly to train due to the complexity of the neural network architecture and the need for more data.

Inference Speed

- **Matrix Factorization:** Faster inference as the learned embeddings are static and can be precomputed.
- **Two-Tower Neural Network:** Slower inference due to the need to transform user features at query time.

Cold-Start Problem

- **Matrix Factorization:** Struggles with cold-start problems, as it relies heavily on user-item interactions.
- **Two-Tower Neural Network:** Can handle new users more easily by leveraging user features and context.

Quality of Recommendations

- **Matrix Factorization:** Not ideal for complex scenarios, as it doesn't explicitly consider user and item features.
- **Two-Tower Neural Network:** Can produce better recommendations by leveraging rich user and item features, leading to more accurate predictions.

In Summary:

- **Matrix Factorization** is a simple and efficient technique, but it may struggle with cold-start problems and complex scenarios.
- **Two-Tower Neural Networks** offer more flexibility and can handle complex relationships between users and items, but they are computationally more expensive to train and infer.

Choosing the Right Model:

The choice of model depends on various factors, including:

- **Data Availability:** If you have a large amount of user-item interaction data, matrix factorization can be a good choice.
- **Complexity of the Problem:** For complex scenarios with rich user and item features, a two-tower neural network may be more suitable.
- **Computational Resources:** Consider the computational cost of training and inference when choosing a model.
- **Real-time Requirements:** If real-time recommendations are required, a faster model like matrix factorization might be preferred.

Evaluation:Offline Metrics

- Precision@k - measures the proportion of videos among top k relevant videos recommended using different values of k.
- mAP (mean Average Precision) - measures ranking quality of recommended videos making it a good fit as our relevance scores are binary.
- Diversity - This metric measures how dissimilar recommended videos are to each other. It is an important metric to track as users are interested to watch diversified videos.
    - To measure diversity - calculate the average pairwise similarity (cosine similarity or dot product) between the videos in that list.
    - Lower average similarity indicates diverse list.
    - Note - Diversity in list is only useful if user finds it useful meaning, we have to use diversity metric with other metrics to ensure relevance and diversity.

Evaluation: Online Evaluation

- Click-through rate (CTR) - $$CTR = \frac{\text{number of clicked videos}}{\text{total number of recommended videos}}$$
    - We cannot measure clickbait videos.

- Number of Completed Videos - shows us how many system recommended videos the user watch.
- Total Watch time - the total time user spent time watching recommended videos.
- Explicit User feedback - The total number of videos that user explicitly liked or disliked. It accurately reflects the user's opinion on recommended videos.

Serving: At serving time, the system recommends the most relevant videos from billions of videos to a given user by narrowing down the selection. So we will use a prediction pipeline which is efficient and accurate at serving requests.
Two issues:

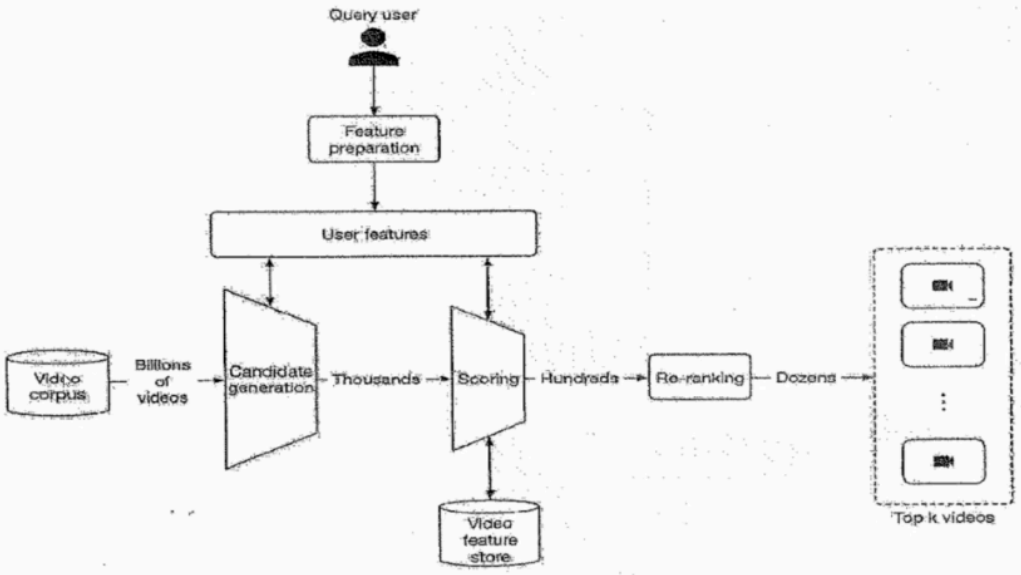
- We are dealing with billions of videos so the serving speed would be slow if we choose a heavy model because it has many features to take as an input.

OR

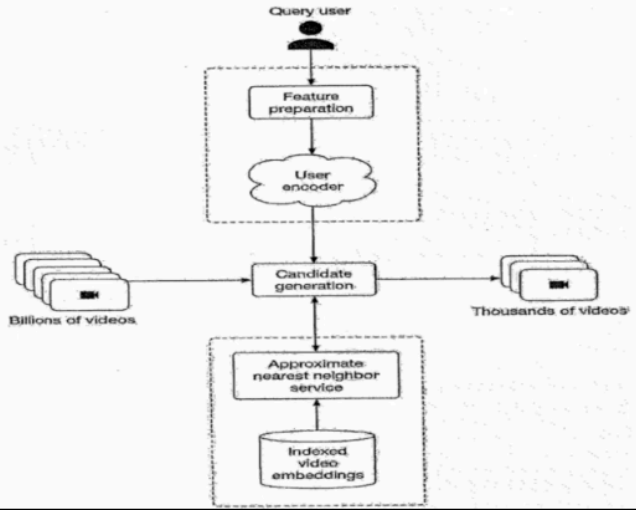
- If we choose a light weight model the recommendations will not be of high quality.

- One solution is to use one model in a multi-stage design:
  - In Stage One, the light weight model narrows down the videos called candidate generation.
  - In Stage Two, the heavier model that accurately scores and ranks the video called scoring.
    We will discuss:
    - 1. Candidate generation
    - 2. Scoring
    - 3. Re-ranking

- Stage One for Candidate Generation.
- Stage Two for Scoring.

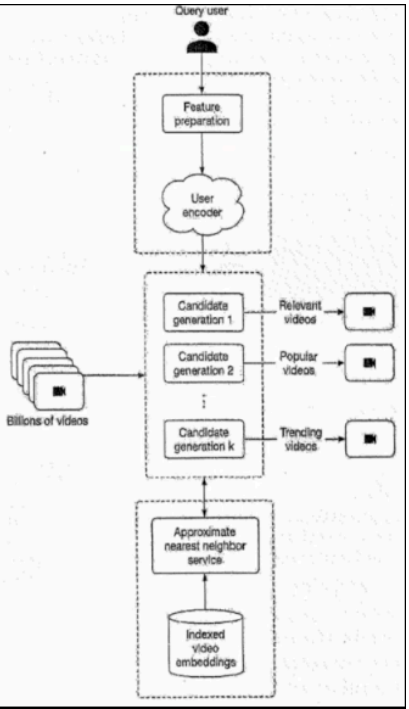


- Candidate Generation: helps us to narrow down the videos from billions to thousands. We focus on efficiently narrowing down the videos than accurately narrowing it down while not being concerned for false positive values.
- For keeping candidate generation fast and handling new users as well, we will choose a model which doesn't rely on video features. So we will use Two-Tower Neural Network.



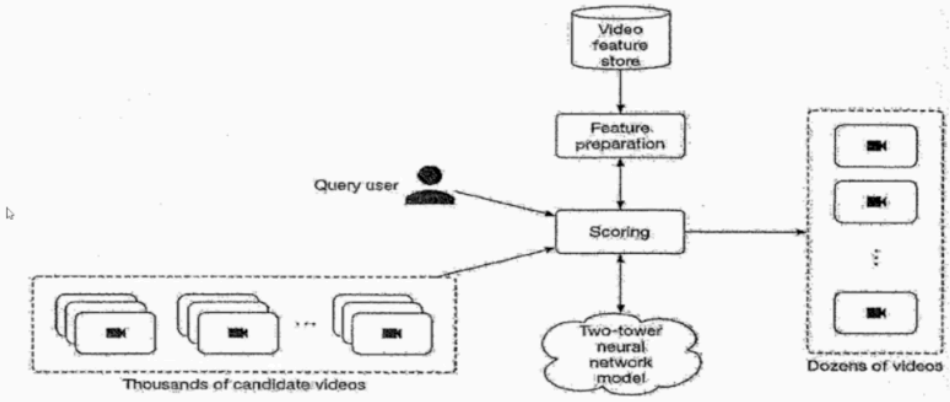
In practice, Companies will choose more than one candidate generation for improved performance of recommendation.

- Users may want to watch videos depending on different reasons, a user may want to watch a video because it is popular/trending or it is relevant to their location. So to include those recommendations it is common to use more than one candidate generation.



Scoring: After candidate generation, videos retrieved are used for scoring also known as ranking, scoring takes the user and candidate videos as input and scores each video to output a ranked list of videos.

- In this stage - We will prioritize accuracy over efficiency so content-based filtering is used which uses video features. We will use Two-Tower Neural Network which is heavier model taking more parameters for ranking the videos.



Re-ranking:
This component re-ranks the videos by adding more criteria or constraints. Eg. We may use a standalone model to determine the “click-bait” videos.
Few things to keep in mind when building re-ranking components.

- Region-restricted videos
- Video freshness
- Videos spreading misinformation
- Duplicate or near-duplicate videos
- Fairness and bias

The Video Recommendation System and tackling its challenges:

Serving Speed - It is important to recommend videos fast. Dealing with billions of videos and recommending them efficiently and accurately can be challenging.

- To address this we used Two Stage design, using light weight model for Stage One for Candidate generation for dealing with billions of videos and narrowing it down to thousands.

Precision - To ensure precision we include heavier model to rank videos which relies on more features including video features.

- Using a heavier model will not affect the serving speed as only a subset of videos are selected after the candidate generation.

Diversity - to ensure diverse recommendation to users we used multiple candidate generations.

Cold-Start problem:
We don't have new user interaction data when they start using the platform.

- For New Users - The predictions are still made using Two-tower Neural Network for new users based on features like age, gender, location, language etc. As the user interacts, we can make better predictions based on new interactions.
- For New Videos - We have the video metadata and content but we don't have video interaction data. To solve this we display the video to random users and collect video interaction data and then fine-tune the model based on new interaction data collected.

Training Scalability:
It is challenging to train models on large datasets in a cost-effective manner. In recommendation systems new interactions are continuously added and the system has to quickly adapt to make accurate recommendations. To quickly adapt to new data, model should be fine-tuned. Our model is based on neural network and can be easily fine-tuned.

Cold-Start Problem

The cold-start problem arises when there is limited or no historical data for new users or new items. This makes it difficult to generate accurate recommendations. The image proposes two strategies to address this issue:

1. New Users:

- **Leverage User Features:** For new users, the system can rely on demographic information like age, gender, location, and language to make initial recommendations.
- **Iterative Learning:** As the user interacts with the platform, the system can collect more data and refine its recommendations.

2. New Items:

- **Content-Based Filtering:** Use item features like genre, director, and keywords to recommend similar items.
- **Exploratory Recommendations:** Introduce new items to a small subset of users and collect their feedback to inform future recommendations.

Training Scalability

Training large-scale recommendation models can be computationally expensive and time-consuming. To address this challenge, the image suggests:

- **Fine-Tuning:** Continuously fine-tune the model on new data to adapt to changing user preferences and item characteristics.
- **Neural Network-Based Models:** Neural networks are highly flexible and can be efficiently trained and updated, making them suitable for large-scale recommendation systems.
- **Distributed Training:** Distribute the training process across multiple machines to accelerate training time.

Additional Considerations:

- **Hybrid Approaches:** Combining collaborative filtering and content-based filtering can help mitigate the cold-start problem.
- **Contextual Bandits:** A reinforcement learning approach that can dynamically adjust recommendations based on user feedback.
- **Transfer Learning:** Leveraging pre-trained models can improve the performance of recommendation systems, especially for cold-start scenarios.