ML System Design for Event Recommendation System

Video:
https://www.youtube.com/watch?v=ULHk2WYM86M&list=PL_b_MRp1BnEj4UuHv1jAGeO1a0VTRXsAk&index=12

Problem Statement:

To build a personalized event recommendation system for an Event Management and Ticketing platform to increase ticket sales.

Clarifying Requirements

1. Clarifying the Requirements:
- To build a personalized event recommendation system to users.
- Business Objective - To increase ticket sales of an Event Management and Ticketing platform.
- Only Event is considered i.e it is announced, event takes place and event is finished meaning event expired.
- We have Description, Price range, location, Date-time etc of event.

- Assumptions and Constraints:
  - We don't have labelled dataset so we will use user interaction data to construct the dataset.
  - Users has allowed to system to share their location, it is a location based event.
  - Users can form peer to peer connections/ become friends.
  - Users can invite others to events.
  - There are both free and paid events.
  - We have 1 million events happening per month and 1 million users across the website/app.
  - Using external APIs Google Maps/Map Services to calculate the travel time.

Key Considerations and Constraints:

1. **Event Lifecycle:**
    - Events have three states: announced, ongoing, and finished.
    - Recommendations should only be made for upcoming events.
2. **Event Data:**
    - Each event has attributes like description, price range, location, and date-time.
3. **User Data:**
    - Users provide location information.
    - Users can form social connections with other users.
    - Users can invite friends to events.
4. **System Scale:**

- The system handles a large number of events and users.
- External APIs are used to calculate travel times.

Potential Recommendation Strategies:

Given the constraints and data available, here are some potential strategies for building a personalized event recommendation system:

1. **Content-Based Filtering:**
    - Recommend events similar to ones the user has previously attended or shown interest in.
    - Use attributes like event category, genre, location, and price range to find similar events.
2. **Collaborative Filtering:**
    - Recommend events that similar users have attended and liked.
    - Utilize user-item ratings or implicit feedback (e.g., event attendance, ticket purchases) to find similar users and their preferences.
3. **Hybrid Approach:**
    - Combine content-based and collaborative filtering to leverage the strengths of both approaches.
    - This can involve creating a hybrid model that considers both user preferences and item similarities.
4. **Contextual Bandits:**
    - Use a reinforcement learning approach to learn the optimal recommendation strategy over time.
    - The system can experiment with different recommendations and learn from user feedback to improve future recommendations.

Additional Considerations:

- **Real-time Recommendations:** Consider using techniques like real-time indexing and retrieval to provide timely recommendations, especially for time-sensitive events.

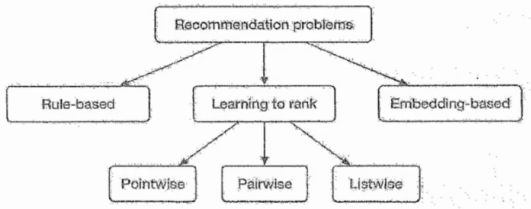
- **Cold-Start Problem:** For new users or new events, leverage content-based filtering or knowledge-based recommendations.
- **User Preferences:** Incorporate user preferences, such as genre, price range, and location, to personalize recommendations.
- **Social Influence:** Leverage user social connections to recommend events that friends have attended or recommended.
- **Evaluation Metrics:** Use appropriate metrics to evaluate the performance of the recommendation system, such as precision, recall, F1-score, and user engagement metrics.

Framing as ML Task

2. Frame the problem as an ML Task:
- Business Objective - Increase the ticket sales.
- ML objective is to increase the number of event registration.

- Specifying the Input/Output:
  - Input - A user.
  - Output - top k events ranked by the relevance to the user.

Choosing the right ML Category: Different ways to solve a recommendation problem
- Recommending Popular events - simple solution.
- Embedding-based models which rely on content-based filtering or collaborative filtering.
- Reformulating it into a ranking problem.



Recommendation Problems

1. **Rule-Based:**
   - Relies on predefined rules to make recommendations.
   - Simple to implement but often lacks personalization.
   - Example: Recommending products based on a user's purchase history.
2. **Learning to Rank:**
   - Trains a model to directly rank items based on their relevance to a user's query or context.

- Can handle complex ranking scenarios and incorporate various features.
- Subdivided into three main approaches:
  - **Pointwise:** Treats each item independently and assigns a score.
  - **Pairwise:** Compares pairs of items and learns to rank them relative to each other.
  - **Listwise:** Considers the entire list of items and optimizes the ranking of the entire list.

3. **Embedding-Based:**
   - Represents items and users as dense vectors in a latent space.
   - Similarity between embeddings is used to make recommendations.
   - Can be based on content-based filtering (recommending similar items) or collaborative filtering (recommending items liked by similar users).

Choosing the Right Approach:

The choice of approach depends on various factors, including:

- **Data Availability:** The amount and quality of available data.
- **Computational Resources:** The complexity of the model and the scale of the dataset.
- **Desired Level of Personalization:** The extent to which recommendations should be tailored to individual users.
- **Real-time Requirements:** The need for real-time or near-real-time recommendations.

In the context of event recommendations, a hybrid approach combining multiple techniques might be effective:

- **Rule-based:** For simple, initial recommendations based on basic criteria (e.g., location, genre).
- **Embedding-based:** For personalized recommendations based on user preferences and item similarities.
- **Learning to rank:** To optimize the ranking of recommended events based on various factors.

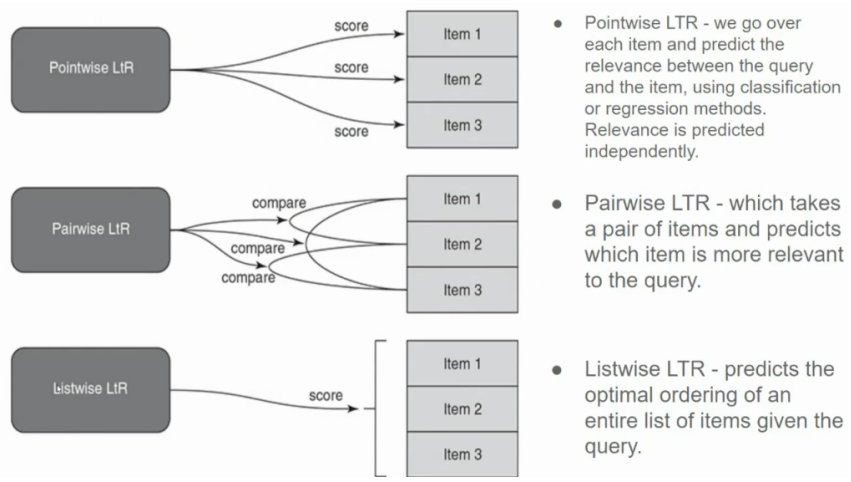
- LTR algorithms : Learning to Rank (LTR) algorithms are used for optimal ordering of items given the query.
  - Having a query, what is the optimal ordering of the list from most relevant item to the least relevant item of the list.
  - Generally, there are three approaches,
    - Pointwise learning.
    - Pairwise learning.
    - Listwise learning.

Learning to Rank (LTR) algorithms are machine learning models designed to automatically rank items, making them especially useful in search engines and recommendation systems. They learn the relevance of items based on training data and optimize the order of results to improve user satisfaction.

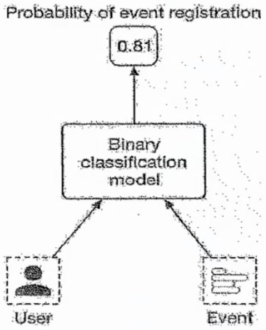
Types of Learning to Rank Approaches

1. **Pointwise**: Treats ranking as a regression or classification problem by assigning a score or class to each item individually. Example: *Linear Regression* or *Support Vector Machines*.
2. **Pairwise**: Considers pairs of items and learns their relative ranking by minimizing the number of incorrectly ordered pairs. Example: *RankNet*, *LambdaRank*.
3. **Listwise**: Directly optimizes the order of a list of items as a whole, typically with a loss function based on ranking metrics like NDCG. Example: *LambdaMART*, *ListNet*.

Each type has trade-offs in terms of computational complexity, accuracy, and applicability depending on the task.



- For simplicity, we will use pointwise approach. As pairwise and listwise do produce more accurate results they are difficult to implement and train.
- So, we will use a binary classification model which takes a single event and predicts if the user will register for it.



Data

Data Preparation:

- Data Engineering: The data available to us is User, Events, Friendship and Interactions.
- User:

ID	Username	Age	Gender	City	Country	Language	Timezone

- Events:

ID	Host User ID	Category/ Subcategory	Description	Price	Location	Date/Time
1	5	Music Concert	Dua Lipa Tour in Miami	200-900	American Airlines Arena Miami, FL	09/18/2022 19:00-24:00
2	11	Sports Basketball	Golden State Warriors vs. Milwaukee Bucks	140-2500	Chase Center SF, CA	09/22/2022 17:00-19:00
3	7	Art Theater	The Comedy and Magic of Robert Hall	Free	San Jose Improv San Jose, CA	09/06/2022 18:00-19:30

- Friendships:

User ID 1	User ID 2	Timestamp when friendship was formed
28	3	1658451341
7	39	1659281720
11	25	1659312942

Data Preparation:
Feature Engineering:

- Event-based recommendations are different from movies or a book.
- They are short-lived as there is less time from event-creation and when it finishes.
- No historical data.
- So, event-based recommendation system faces cold-start problem and constant new-item problem.
- To solve these issues we will be creating as many useful features possible.
- Features created will be of following categories:
  - Location-related features.
  - Social-related features.
  - Time-related features.
  - User-related features.
  - Event-related features.
- In practise, features can be much higher.

Data Preparation : Feature Engineering (Cont.)

- **Location-related feature:**
  - **How accessible is the event?**
    - Walk score - how walkable is the event is based on the distance to nearby amenities. Walk score between 0-100.
    - Walk score similarity - The event's score and the user's average walk score registered by the previous events registered by user.

Category	Walk score	Description
1	90-100	No car needed
2	70-89	Very walkable
3	50-69	Somewhat walkable
4	25-49	Car-dependent
5	0-24	Requires a car

- Similar to walk score we can have transit score, transit score similarity, bike score, bike score similarity etc.

Data Preparation : Feature Engineering - Location-related features.

- **Is the event in the same country and city as the user?**
  - A very important factor for a user is whether the event is located in the same country and city as they are located. We can create two features here:
    - If the user's country is same as event's country then the feature is 1, otherwise 0.
    - If the user's city is same as event's city then the feature is 1, otherwise 0.
- **Is the user comfortable with the distance?**
  - Some users prefer shorter distance for event while some prefer events that are further away.
  - We use following feature to capture this - distance between user's location and event's location, this value can be obtained from APIs and can be bucketized into categories.
    - 0 : less than a km.
    - 1 : 1 - 5 km.
    - 2 : 5 - 20 km.
    - 3 : 20 - 50 km.
    - 4 : 50 - 100 km.
    - 5 : 100+ km.

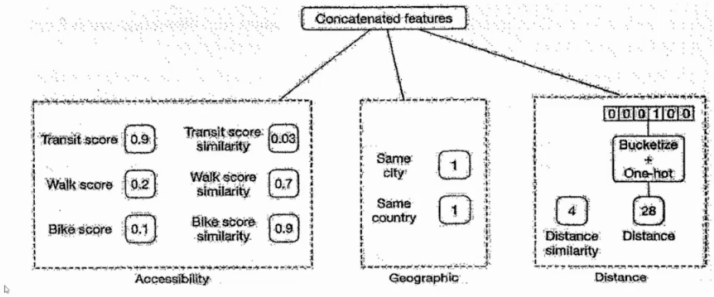
Why These Features Matter:

These features are crucial for personalized event recommendations:

- **Relevance:** Users are more likely to be interested in events that are geographically relevant to them.
- **User Preferences:** By considering distance preferences, the system can tailor recommendations to individual users.
- **Feasibility:** These features can help filter out events that are impractical for the user to attend based on distance and location.

By incorporating these features into a recommendation model, we can improve the accuracy and relevance of the recommendations, leading to a better user experience.

- Distance similarity: Difference between the distance to an event and the average distance (in reality, median or percentile range can be used) previously registered by the user.

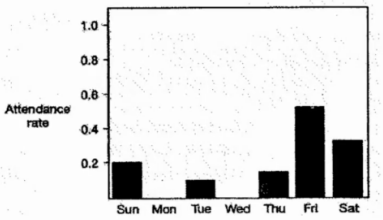


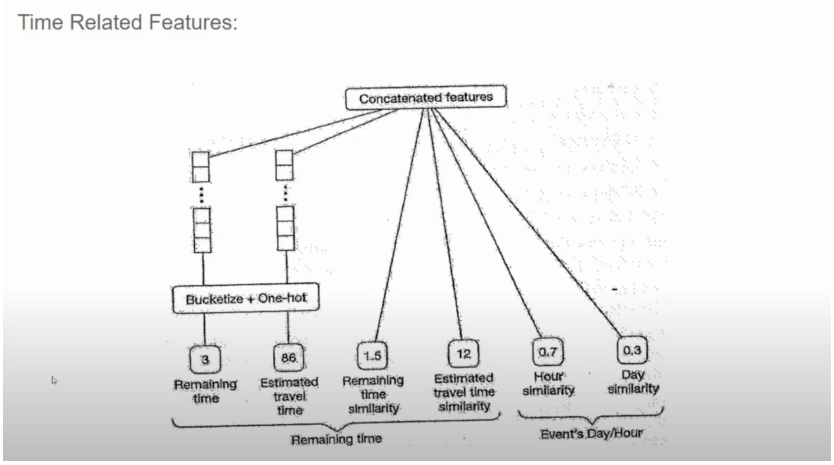
Data Preparation : Feature Engineering
Time-related features:

- **How convenient is the time until an event?**
  - Some users plan the event days in advance and some don't, so let's create feature to capture this:
  - 0 : less than 1 hr remaining till the event starts.
  - 1 : 1- 2 hours
  - 2 : 2- 4 hours
  - 3 : 4 - 6 hours
  - 4 : 6 - 12 hours
  - 5 : 12 -24 hours
  - 6 : 1 -3 days
  - 7 : 3 -7 days
  - 8 : 7+ days
- Remaining time similarity : difference between "remaining time" and average "remaining time" of events previously registered by the user.
- Estimated travel time from the user's location to the event's location -from external services and bucketized into categories.
- Estimated travel time similarity : The difference between the estimated travel time to the event in question, and the average estimated travel time of event's previously registered by the user.

Are the date and time convenient for the user?

- Some users prefer weekdays and some users prefer weekends for the events.
- To capture this we create a user profile which is a vector of size 7 and each value counts the number of events the user attended on a particular day. By dividing these values with the total number of attended events we get the historical rate of event attendance for each day of the week.
- Similarly, we can create per-hour user profiles.
- Similarly, we add day and hour similarity.





Key Features and Their Engineering:

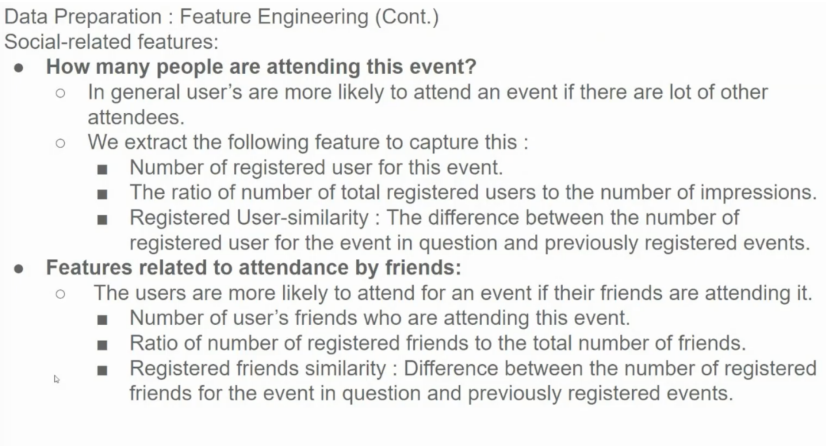
1. **Remaining Time:**
    - **Bucketization:** The remaining time to the event can be categorized into buckets like "less than a day," "1-3 days," "3-7 days," and "more than 7 days."
    - **One-hot Encoding:** Each bucket can be represented as a binary feature.
2. **Estimated Travel Time:**
    - Similar to remaining time, estimated travel time can be bucketed into categories like "less than an hour," "1-2 hours," "2-4 hours," and so on.
    - **One-hot Encoding:** Each bucket can be represented as a binary feature.
3. **Time Similarity:**
    - This feature captures the similarity between the user's preferred event times and the event's scheduled time.
    - It can be calculated using techniques like cosine similarity or Euclidean distance between time vectors.
4. **Event's Day/Hour:**
    - This feature captures the specific day and hour of the event.
    - It can be represented as categorical features or numerical features (e.g., hour of the day).

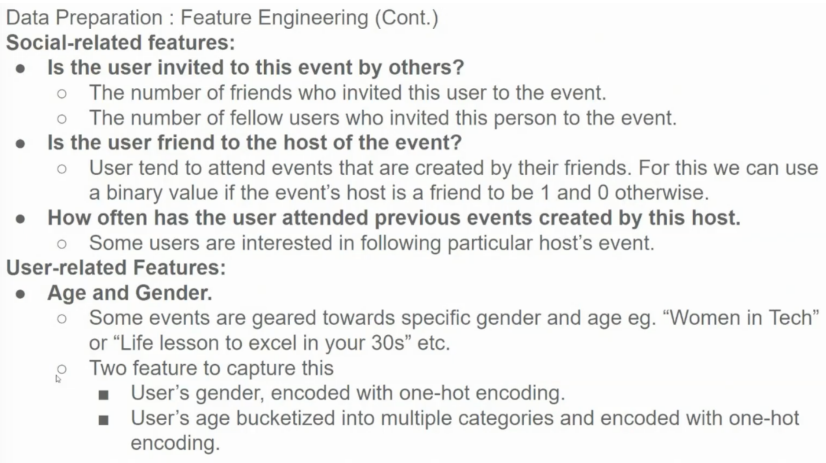
Concatenated Features:

These features can be combined or concatenated to create more complex features. For example, combining "remaining time" and "estimated travel time" can create a feature that captures the overall urgency of the event.

Why These Features Matter:

- **Timeliness:** Users are more likely to be interested in events that are happening soon or within their preferred time frame.
- **Convenience:** Users may prefer events that are easier to attend, considering factors like travel time and event duration.
- **Personalization:** By understanding a user's time preferences, the system can tailor recommendations to their specific needs.





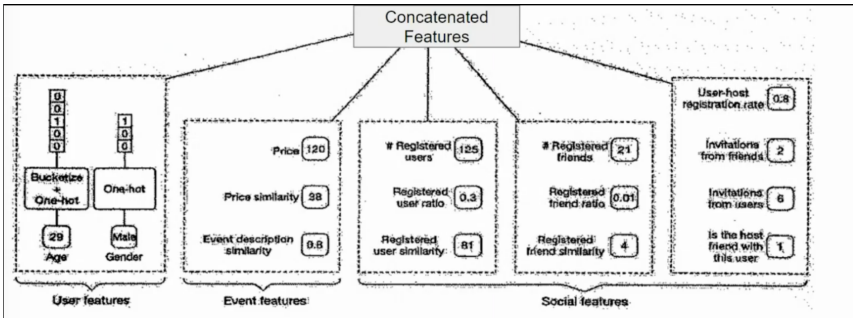
Data Preparation : Feature Engineering (Cont.)

Event-related features:

Price of an event

Price of an event might affect the user's decision to register it. Some features are:

- Event's price, bucketized into few categories. Eg.
  - 0 : Free.
  - 1 : Rs. 1 - 500
  - 2 : Rs. 500 - 1000
  - 3 : Rs. 1000 - 2000
  - 4 : Rs. 2000 - 4000
  - 5 : Rs. 4000+
- Price similarity - Difference between the price of the event in question and the average price of events previously registered for by the user.
- **How similar is the event's description to the previously registered event's description.**
  - This indicates that user might be interested in specific events, eg if the word "concerts" have been in previous registrations, it means that user is interested in concerts.
  - To capture this we use TF-IDF and to calculate distance use cosine distance and also check the performance of model being trained with and without TF-IDF.



Additional Features:

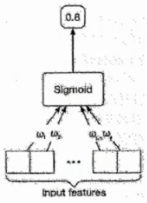
- Host's popularity, user's search history, event's category, auto generated event tags etc.

Additional points to be considered:

- **Batch (static) vs Streaming (dynamic) features** : Batch features are features that change less frequently eg. gender, age , event description etc and streaming features are features that change quickly such as number of users registered or time remaining until event etc.
- **Feature computation efficiency** : Computing features in real-time is not efficient. Eg. instead of computing user's location and the event's location we can use two separate features and let model implicitly compute those information from two locations.
- **Using a decay factor** - is how long the user's actions can be referred for it's optimal use as user's habits changes i.e. using user's last X interactions. Decay factor gives more weightage to user's more recent interactions/behaviours.
- **Using embedding learning** to convert user and event into embedding vectors and used for representing them.
- **Creating features from user's attributes may create bias** - eg using age and gender for finding candidate a good match for a job can lead to discrimination. That is the reason to be vary of user's features as it can introduce bias.

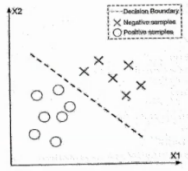
Model Development:
Binary classification models:

1. **Logistic Regression** - models the probability of binary outcome by using a linear combination of one of more features.



Pros:

- Fast Inference speed.
- Efficient training.
- Works well with the data is linearly separable (diag. on right).
- Interpretable and easy to understand.



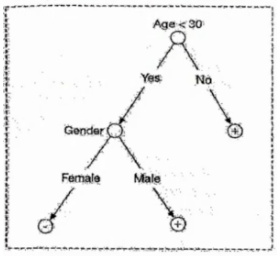
Linearly separable features

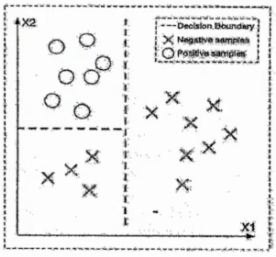
Cons:

- Non linear problems cannot be solved since it uses linear combination of input features.
- Multicollinearity occurs when independent variables in a regression model are correlated. This correlation is a problem because independent variables should be independent. If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results.

2. Decision Tree:

Diagram 1 - Decision tree
Diagram 2 - Learned Decision Boundary





Pros:

- Fast Training.
- Fast Inference.
- Little to no data preparation.
- Interpretable.

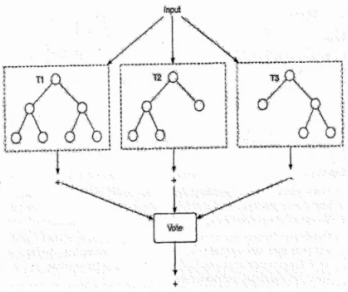
Cons:

- Non optimal decision boundary.
- Overfitting.

In practise, naive decision trees are rarely used because they are too sensitive to change in input data (variance). Two common techniques used to reduce the sensitivity of decision trees are:

- Bootstrap AGGregatING (Bagging).
- Boosting.

- Bagging:



Advantages of Bagging:

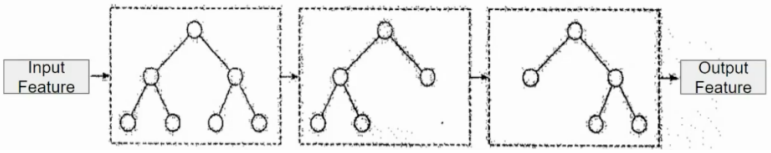
- Reduces the effect of overfitting.
- Does not affect the training time much as the models are parallely trained.
- Because of models being trained in parallel, it can process input in parallel hence less latency added at inference time.

Even for it's advantages Bagging is not useful when model faces underfitting.

Model Development (Cont):

Boosting:

- The key property of boosting ensembles is the idea of correcting prediction errors. The models are fit and added to the ensemble sequentially such that the second model attempts to correct the predictions of the first model, the third corrects the second model, and so on.
- This typically involves the use of very simple decision trees that only make a single or a few decisions, referred to in boosting as weak learners. The predictions of the weak learners are combined using simple voting or averaging, although the contributions are weighed proportional to their performance or capability. The objective is to develop a so-called "strong-learner" from many purpose-built "weak-learners."



Pros:

- **Boosting reduces bias and variance.**

Cons:

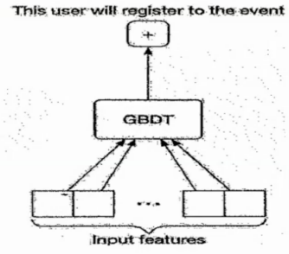
- **Slower training and inference time.**

Boosting is preferred more as it reduces bias and variance both.

- Some of the boosting-based decision trees are XGBoost, Adaboost and Gradient Boost commonly used to train classification models.

3. **Gradient-Boosted Decision Tree (GBDT)**:

- GBDT is one of the commonly used tree-based model utilizing Gradient Boost to improve decision trees. Variants of GBDT, XGBoost has also demonstrated strong performance.



Pros:

- **Easy data preparation.**
- **Reduce variance.**
- **Reduce Bias.**
- Works well within the structured data.

Cons:

- **Lots of Parameters to tune** - Number of iterations, tree depth, regularization parameters etc.
- **Does not work well with Unstructured data.**
- **Unsuitable for continual learning from streaming data.**

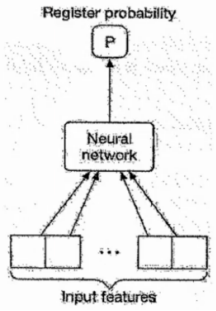
Continuing with the unsuitability for continual learning:

- In the event recommendation system - the model has to continuously learn about new data and new/changing behaviours of users.
- New data and new user interactions are continuously available to the model and is vital to adapt to the changing behaviour of the users and the data.
- This is not possible without continuous learning. We need to train GBDT from scratch regularly which is very expensive.

4. Neural Network:

In event-recommendation system, we have many features that might not correlate linearly with the outcome and have to learn complex relationships. Also we use continual learning to adapt to new data.

- All the complexities and issues can be addressed by using Neural Networks.



Pros:

- Continual Learning.
- Works well with unstructured data.
- Expressiveness.

Cons:

- Computationally expensive to train.
- The quality of input data strongly influences the outcome.
- Large training data.
- Black-box nature - meaning it is not easy to understand which features are influencing on outcomes as input feature goes through multiple layers of outcome.

Which model to use?

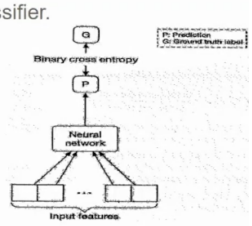
- We can start with GBDT's variant XGBoost which is fast to implement and train. The result can serve as an initial baseline.
- Once we have a baseline we can shift to building models with NNs:
  - Massive training data is available due to user interaction, inviting friends, new events being registered. Given the number of users it creates big dataset. NN can work with non-linear data.
  - They are good with complex tasks and taking many parameters.

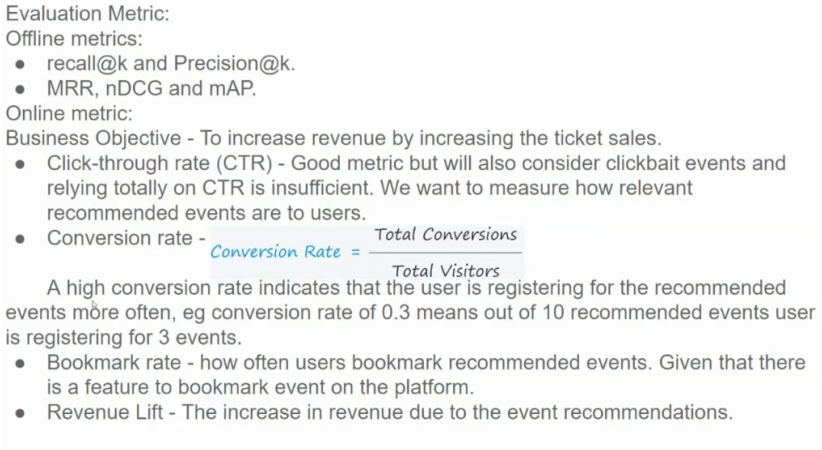
Model Training: Constructing the dataset.

- Extracted <user,event> pair from interaction data,
  - If user attended/registered the event it is labelled as 1 and 0 otherwise.

#	Extracted ⟨user, event⟩ features							Label
1		1	0	1	1	0	1	1
2		0	0	0	1	1	0	0

- One issue is that, the dataset set will be very noisy making an imbalanced dataset.
  - To solve this we can use,
    - Focal loss or class-balanced loss to train the classifier.
    - Undersample the majority class.
  - Choosing the loss function:
    - Binary cross-entropy to optimize NN model.





Offline Metrics

These metrics are used to evaluate the model's performance on a static dataset.

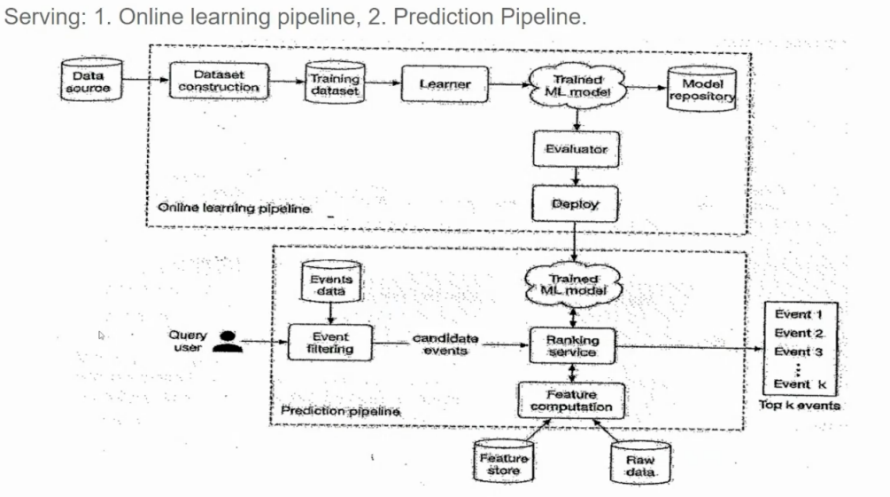
- **Recall@k and Precision@k:**
    - **Recall@k:** Measures the proportion of relevant items that are recommended within the top k recommendations.
    - **Precision@k:** Measures the proportion of recommended items that are actually relevant within the top k recommendations.
    - These metrics help assess the system's ability to retrieve relevant items and avoid recommending irrelevant ones.
- **MRR (Mean Reciprocal Rank):**
    - Measures the average reciprocal rank of the first relevant item in the ranked list.
    - A higher MRR indicates better performance, as it rewards models that rank relevant items higher.
- **nDCG (Normalized Discounted Cumulative Gain):**
    - Considers the position of relevant items in the ranked list.
    - It assigns higher weights to items ranked higher and penalizes irrelevant items.
    - A higher nDCG indicates better ranking quality.
- **MAP (Mean Average Precision):**
    - Measures the average precision at different recall levels.

    - It considers both precision and recall, making it a comprehensive metric.

Online Metrics

These metrics are used to evaluate the model's performance in a real-world setting.

- **Click-Through Rate (CTR):**
    - Measures the proportion of users who click on a recommended event.
    - While CTR is a useful metric, it's important to consider the quality of the clicks. A high CTR could be due to clickbait events rather than truly relevant recommendations.
- **Conversion Rate:**
    - Measures the proportion of users who convert after clicking on a recommended event (e.g., purchasing a ticket).
    - A higher conversion rate indicates that the system is effectively recommending events that users are interested in.
- **Bookmark Rate:**
    - Measures the proportion of users who bookmark recommended events.
    - This metric can be used to assess the long-term impact of recommendations, as bookmarked events may be considered for future purchases.
- **Revenue Lift:**
    - Measures the increase in revenue generated by the recommendation system compared to a baseline (e.g., no recommendations).
    - This is the ultimate metric for evaluating the business impact of the system.



1. **Online Learning Pipeline:** Responsible for continuously training and updating the recommendation model.
2. **Prediction Pipeline:** Responsible for generating recommendations for users.

Online Learning Pipeline:

1. **Data Source:** New events and user interactions are continuously fed into the system.
2. **Dataset Construction:** Relevant features are extracted from the raw data to create a structured dataset for training.
3. **Training Dataset:** The constructed dataset is used to train the machine learning model.
4. **Learner:** The machine learning model (e.g., a neural network, decision tree, or ensemble model) is trained on the dataset.
5. **Trained ML Model:** The trained model is stored in a model repository.
6. **Evaluator:** The model's performance is evaluated on a validation dataset to assess its accuracy and effectiveness.
7. **Deploy:** The trained model is deployed to the prediction pipeline.

Prediction Pipeline:

1. **Query:** A user submits a query (e.g., search term, location, interests).
2. **Event Filtering:** The system filters events based on the user's query and preferences.

3. **Candidate Events:** A set of candidate events is generated based on the filtering criteria.
4. **Feature Computation:** Features are extracted from the candidate events and user profile to create feature vectors.
5. **Ranking Service:** The trained machine learning model ranks the candidate events based on their feature vectors and the user's query.
6. **Top k Events:** The top k events are selected from the ranked list and presented to the user as recommendations.

Key Components and Their Roles:

- **Data Source:** Provides a continuous stream of data to train and update the model.
- **Feature Engineering:** Extracts relevant features from raw data to improve model performance.
- **Machine Learning Model:** Learns patterns from the data to make accurate predictions.
- **Model Deployment:** Deploys the trained model to a production environment.
- **Event Filtering:** Reduces the search space by filtering out irrelevant events.
- **Feature Computation:** Prepares the data for the ranking model.
- **Ranking Service:** Ranks events based on their relevance to the user's query and preferences.

Serving

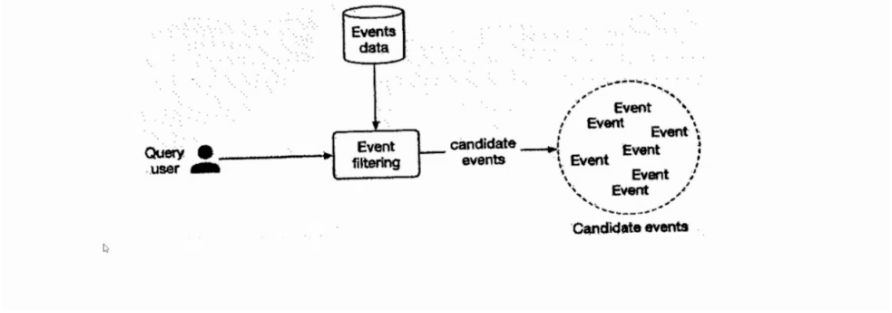
Online learning Pipeline:

- Because event recommendation system are cold-start and faces constant new-item problem.
- Consequently, continuously fine-tuned to adapt new data.
- So, Online learning Pipeline is responsible training new models by incorporating new data, evaluating trained models and deploying them.

Prediction Pipeline:

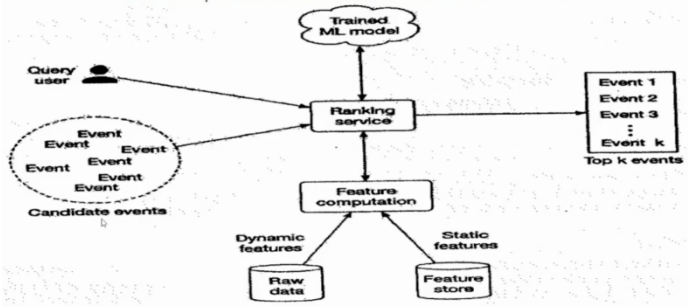
- Responsible for recommending top k most relevant events to a given user.

Event Filtering: Helps us to narrow down from millions of results to a small subsets of events. We can have candidate events from event filtering.

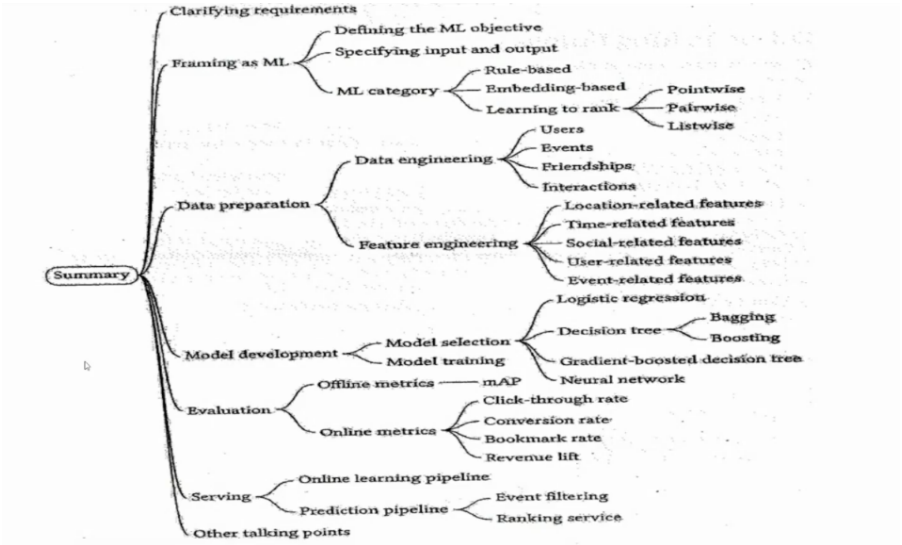


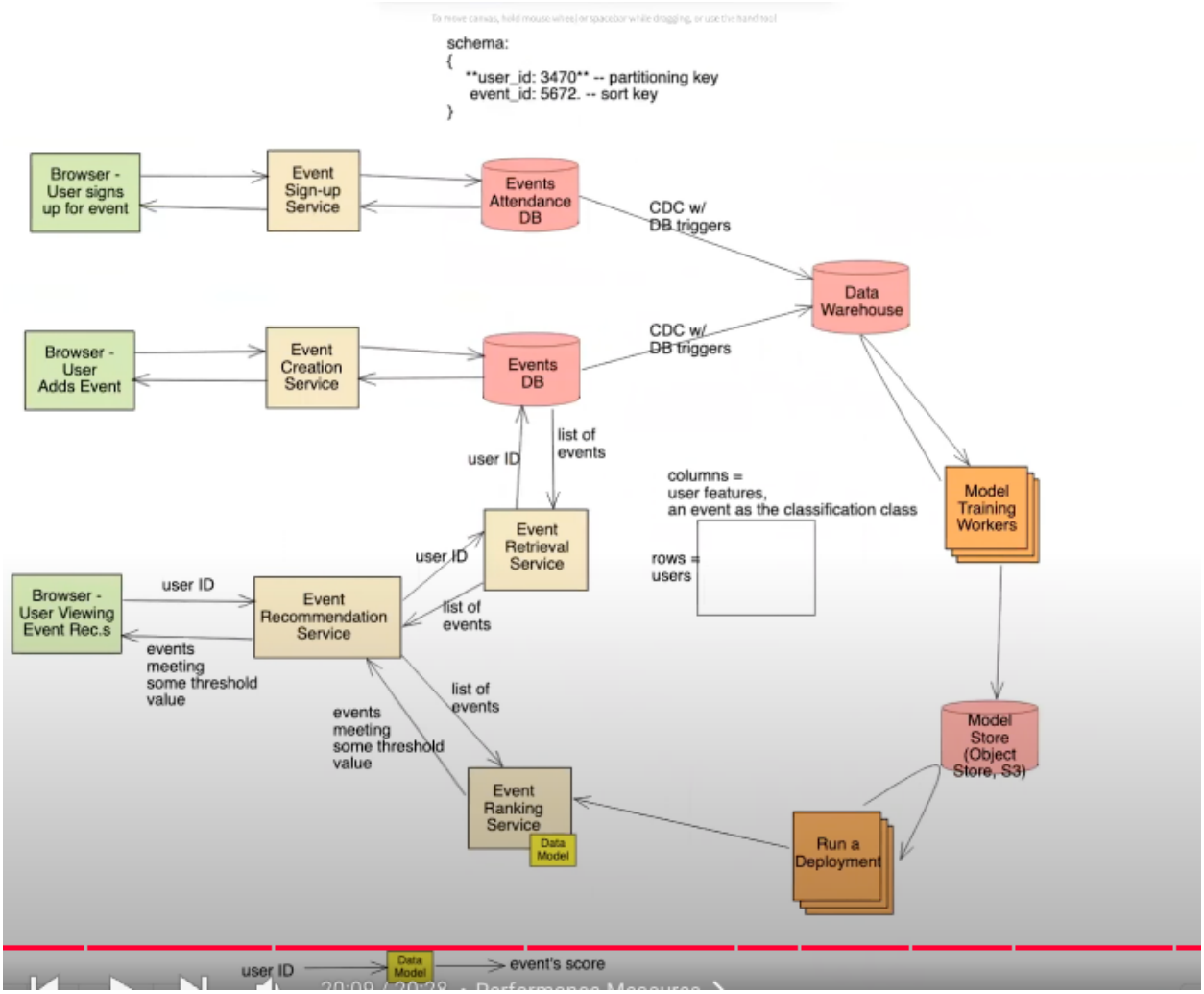
Ranking Services:

- Takes <user,event> pair as input from the filtering component.
- Sorts the events based on the probability predicted by the model and outputs a ranked list of top k most relevant events to the user.



-Ranking services interacts with feature computation component responsible for computing features that the model expects.
-Static features are obtained from feature store and dynamic features in real-time from raw data.





https://www.youtube.com/watch?v=1yLz6uvUJCQ&list=PLlvnxKilk3aKx0oFua-HTtFf-d_inQ8Qn&index=6