ML System Design People You May Know

1. Business Objective

The goal of the "People You May Know" (PYMK) feature is to recommend potential connections (friends, colleagues, etc.) to users based on their network and interests. For platforms like Facebook or LinkedIn, these recommendations help increase user engagement by encouraging more connections, improving user experience, and ultimately contributing to increased time spent on the platform. It also helps with network growth.

Specific Business Objectives:

- Increase user engagement (e.g., more connection requests, more interactions with recommended profiles).
- Improve the relevance of recommendations to increase the likelihood of users connecting with suggested profiles.
- Enhance platform activity, resulting in improved user retention and overall growth.

2. Clarifying Questions

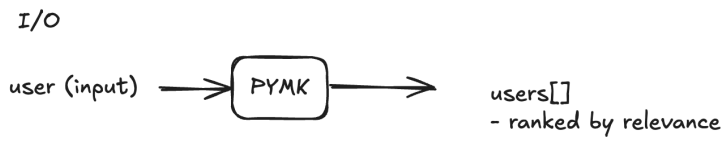
Before framing the problem, we need to gather additional details:

- **User behavior data:** What data do we have on users' interactions (likes, comments, profile views, etc.)?
- **Data availability:** Do we have data on mutual connections, interests, and demographics? Are there any constraints on how long a user has been active on the platform?
- **Relevance:** How do we define relevance for the PYMK feature? Is it based purely on mutual connections, or should interests, past interactions, and other metadata play a role?
- **Negative feedback:** Do we account for users' dismissals of recommendations? Do users often decline or ignore these recommendations, and how do we handle that?
- **Evaluation frequency:** How often do we need to update the recommendations for users? Real-time or batch?
- Requirements - Scalability: 1 B total users, on avg. 1000 connection per user.
- Assumptions: symmetric friendships.

3. Framing as an ML Problem

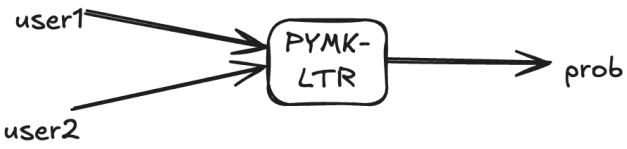
This can be framed as a **recommendation system** problem, where the objective is to predict which users are the most relevant (i.e., the most likely to connect with a given user) based on their profile and behaviors.

I/O: I: user_id, O: ranked list of recommended users sorted by the relevance to the user



This problem can be solved via following:

- **Collaborative Filtering:** Recommending people based on the historical behavior of users (e.g., mutual connections, interactions).
- **Content-Based Filtering:** Recommending people based on profile attributes, interests, or activities (e.g., job titles, skills, shared groups).
- Ranking problem: pointwise LTR - binary classifier (user_i, user_j) -> p(connection) cons: doesn't capture social connections



-
- Additionally, we can use **graph-based models** (e.g., graph neural networks) to model relationships between users, taking into account the network structure. This is what we can use here.

4. Data and Feature Engineering

User Data:

- **Profile Data:** Age, gender, location, job title, education, etc.
- **Behavioral Data:**
    - Interactions (comments, likes, shares)
    - Past connections (friends, colleagues, etc.)

    - Messages or group memberships
    - Search history and content viewed
- **Social Network Data:**
    - Graph structure (users and their connections)
    - Number of mutual connections
    - Recent connections made by the user
    - Engagement history with specific people (e.g., profile visits)

Feature Engineering:

- User Features:
    - skills, groups, job titles
    - Demographics (Age, gender, location)
    - Account/Network info: No of connections, followers, following, requests, etc, account age, which groups the user is in, interactions in those groups
    - Historical interaction features (e.g., how often a user interacts with a group that the recommended user is part of)
    - Interaction history (No of likes, shares, comments)
    - Context (device, time of day, etc)
- User-user connections:
    - Connection: IDs(user1, user2), connection type, timestamp, location
    - edu and work affinity: major similarity, companies in common, industry similarity, etc
    - social affinity: No. mutual connections (time discounted)
- User-user interactions: IDs(u user1, user2), interaction type, timestamp
- Negative Feedback Data:
    - If available, track whether users reject or dismiss recommendations.

5. Possible ML Models

Baseline Model:

A simple **Collaborative Filtering** model (e.g., **Matrix Factorization** or **SVD**):

- Uses user-item interaction history (e.g., connection requests, mutual connections) to make predictions based on similar behavior.

Pros:

1. Simple to implement and provides a quick baseline.
2. Works well for users with rich interaction history.

Cons:

- Might not scale well for new users (cold-start problem).
- Doesn't utilize detailed user features beyond connections.

Final Model:

Similar to other ranking models, a good idea for a PYMK algorithm is a 2-stage modeling approach, consisting of candidate selection (first stage) and candidate ranking (second stage). The first stage filters down the entire user population to a more manageable number of candidates (e.g. from 1 billion to few Thousand), and the second stage ranks these candidates by connection probability, i.e. how likely it is that the two users are going to connect.

1. **Candidate selection**: Perhaps the simplest and also most intuitive heuristic for candidate selection is 'friends-of-friends' (FOF): consider all second-degree connections as possible PYMK candidates. It turns out that this heuristic is in fact extremely powerful: a 2010 public slide deck from Facebook explains that 92% of all new connections are from one of the (on average) 17K FOFs of a user. In other words, FOF candidate selection cuts down the PYMK candidate pool by 6 orders of magnitude (assuming the entire user population is in the Billions), while still retaining 92% recall. LinkedIn uses FOF candidate selection as well, and they combine it with additional heuristics such as selecting all connections of co-workers as well.

Graph Neural Networks (GNNs) or **Graph Convolutional Networks (GCNs)**: These models are excellent for recommendation tasks involving structured data, where the relationships between entities (users, connections, etc.) are important.

2. **Ranking.** We can rank PYMK candidates (extracted in the first stage) with a **binary classification model**, where the rank is simply the model score itself: the higher the score, the higher the probability of a connection. During offline model training, we

can train the model to predict existing connections between users. Then, at inference time, we can score all PYMK candidates for a user, and display the top N candidates from that list.

- what are our nodes and edges

  - node: each node is a person
  - edge: connection
  - symmetric connection

There are three types of predictions that can be defined:
  - Graph level prediction: lets say we model a chemical compound and if it is toxic or not.
  - node level prediction: making a prediction about a node whether bad actor or spammer. on a single node
  - edge level prediction: predict an edge is present between two nodes or if two users are likely to connect.

Three main entities
- user
- connection (between the users, like friendship)
- interactions

user table
  user id - city - school - age - etc

connection table
  user id 1 - user id 2 - time they got connected

interactions table
  user 1 liked/commented/ profile view on another user's profile

Feature Engineering

user specific
- if account created recently, then might not be suggested
- number of connections, followers, number of reactions (likes/comments)

user-user
- education and work/company, city
- number of common/mutual connections

Where do we use these features: each node in the graph, concacetante these users

Training

embeddings are close to each other for connections

We need to measure similarity between users. Loss: distance between connected users

GraphSage, GCN can be used (https://medium.com/@patwei/using-graph-neural-networks-to-predict-analyze-links-in-friendship-networks-5d

Steps:
1. At time t, we have a snapshot of the graph and we are trying to predict what might happen at time t+delta
2. Compute initial node and edge features of the graph
   similar to embedding mentioned earlier
3. Creating labels
   connected or not connected

For a "People You Might Know" system using a Graph Neural Network (GNN), the **loss function** typically evaluates the similarity between predicted connections and ground truth. Common

In order to train such a binary classifier, we need a rich and diverse set of features. These can include:

- the number of common connections,
- whether the two users work at the same employer, or have worked at the same employer in the past,
- whether the two users went to the same school,
- the existence of any common interests, groups, skills, or hobbies,
- whether the two users have a common job title,
- the physical distance between the user's locations,
- the semantic similarity of the text content that two users are posting (using some form of text embedding), or
- whether the two users are tagged together in posts.
- whether the two users liked each other's posts.

6. Evaluation Metrics

Offline Metrics:

- **Precision and Recall:** Measures how well the model identifies relevant recommendations.
- **Mean Average Precision (MAP):** Measures ranking quality of recommendations.
- **Mean Reciprocal Rank (MRR):** Measures the rank of the first relevant recommendation.
- **AUC (Area Under the ROC Curve):** Measures model's ability to distinguish between relevant and irrelevant recommendations.
- **F1 Score:** Harmonic mean of precision and recall.

Online Metrics:

- **Click-Through Rate (CTR):** Tracks how often users click on a recommended connection.

- **Connection Acceptance Rate:** Measures the percentage of recommendations that lead to successful connections.
- **Engagement Rate:** Tracks how often users engage with newly connected users (messages, comments, likes).
- **Dwell Time:** Measures the time spent on profile pages of recommended users.
- Connection Add Rate: Ratio of user clicks on PYMK suggestions to total PYMK shown.
- Successful Session Rate: Time spent browsing PYMK suggestions to find connections.
- Counter Metrics: Track negative user feedback on PYMK suggestions (e.g., hiding profiles, reporting inappropriate content).

7. Deployment and Serving

- **Batch Serving:** For large-scale systems, recommendations can be generated periodically (e.g., once a day or week) using batch processing pipelines.
- **Real-time Serving:** For more personalized and dynamic recommendations, real-time serving systems (e.g., using **TensorFlow Serving** or **TorchServe**) could be implemented. This is important when recommendations need to adjust based on the most recent user activity.

Deployment Stack:

- **Model Serving Infrastructure:** Cloud-based infrastructure such as AWS SageMaker, GCP AI Platform, or custom Kubernetes setups for serving models at scale.
- **Data Pipeline:** Use tools like **Apache Kafka** or **Google Cloud Pub/Sub** to stream real-time user interactions to the recommendation system for quick updates.

8. Monitoring and Infrastructure

- **Model Performance Monitoring:** Use tools like **Prometheus** and **Grafana** to track model metrics (CTR, engagement rates, etc.) in real-time.
- **Drift Detection:** Continuously monitor model performance and detect any degradation in metrics due to changing user behavior (e.g., **Concept Drift**).

- **Logging and Error Tracking:** Use centralized logging systems like **Elasticsearch** or **Splunk** for tracking system errors, slowdowns, or failures in recommendation serving.