Recommendation System – Two-Tower Model

Version: 1.0
File: rp_two_tower_network.py
Author: Bhakti Ayarekar
Updated: 20 October 2025

---

1. Problem Statement

We want to build Recommendation System based on users and video(post) from keek's platform. Traditional recommendation is based on filter+sorting methods, we need to build personalised system. (Traditional method - Show me all videos matching conditions, sorted by entry_date and country.)

On What we are working: (Con's of SQL)

- Treats all users the same within a rule set
- Can't *learn* preferences (e.g. "I like funny Hindi clips but not sports")
- Relies on hard-coded conditions (fixed thresholds, no adaptation)
- Doesn't improve with user behavior

2. Dataset Used :

1) From Production Postgress SQL used table - stream_viewer, users, posts

2) In my project I used query on production dataset :

For Interaction Table - On Streamer viewer

```
SELECT
   user_id,
   post_id,
   SUM(likes) AS likes,
   SUM(view_count) AS views,
   SUM(CASE WHEN saved = 'Y' THEN 1 ELSE 0 END) AS saves
FROM
   public.stream_viewer
WHERE
   post_id IS NOT NULL
   AND likes IS NOT NULL
   AND likes <> 0
GROUP BY
   user_id,
   post_id
ORDER BY
   user_id, post_id;
```

For User Table - On User table

```
SELECT DISTINCT
   u.id AS user_id,
   u.country
FROM
   users u
INNER JOIN (
   SELECT DISTINCT user_id
   FROM public.stream_viewer
   WHERE
      post_id IS NOT NULL
      AND likes IS NOT NULL
      AND likes <> 0
) sv
   ON u.id = sv.user_id
WHERE
   u.country IS NOT NULL
   AND u.country <> ''
ORDER BY
   u.id;
```

For Post Table- On Post table

```
SELECT
   p.post_id,
   p.user_id AS post_owner_id,
   COALESCE(
      NULLIF(
         CASE
            WHEN p.country IS NULL OR p.country = '' OR p.country = 'default'
               THEN u.country
            ELSE p.country
         END,
         ''
      ),
      'IN'
   ) AS effective_country,
   p.lang
FROM
   public.post p
LEFT JOIN (
   SELECT DISTINCT ON (u.id)
      u.id AS user_id,
      u.country
   FROM users u
   INNER JOIN (
      SELECT DISTINCT user_id
      FROM public.stream_viewer
      WHERE
         post_id IS NOT NULL
         AND likes IS NOT NULL
         AND likes <> 0
   ) sv
```

```
        ON u.id = sv.user_id
    WHERE
        u.country IS NOT NULL
        AND u.country <> ''
) u
    ON u.user_id = p.user_id
WHERE
    p.post_id IN (
        SELECT DISTINCT post_id
        FROM public.stream_viewer
        WHERE
            post_id IS NOT NULL
            AND likes IS NOT NULL
            AND likes <> 0
    )
ORDER BY
    p.post_id;
```

Then, After Query

Data Tables look like this :

Interaction Table:

user_id	post_id	likes	Views	Saves
1001	1401	1	1	0

User Table:

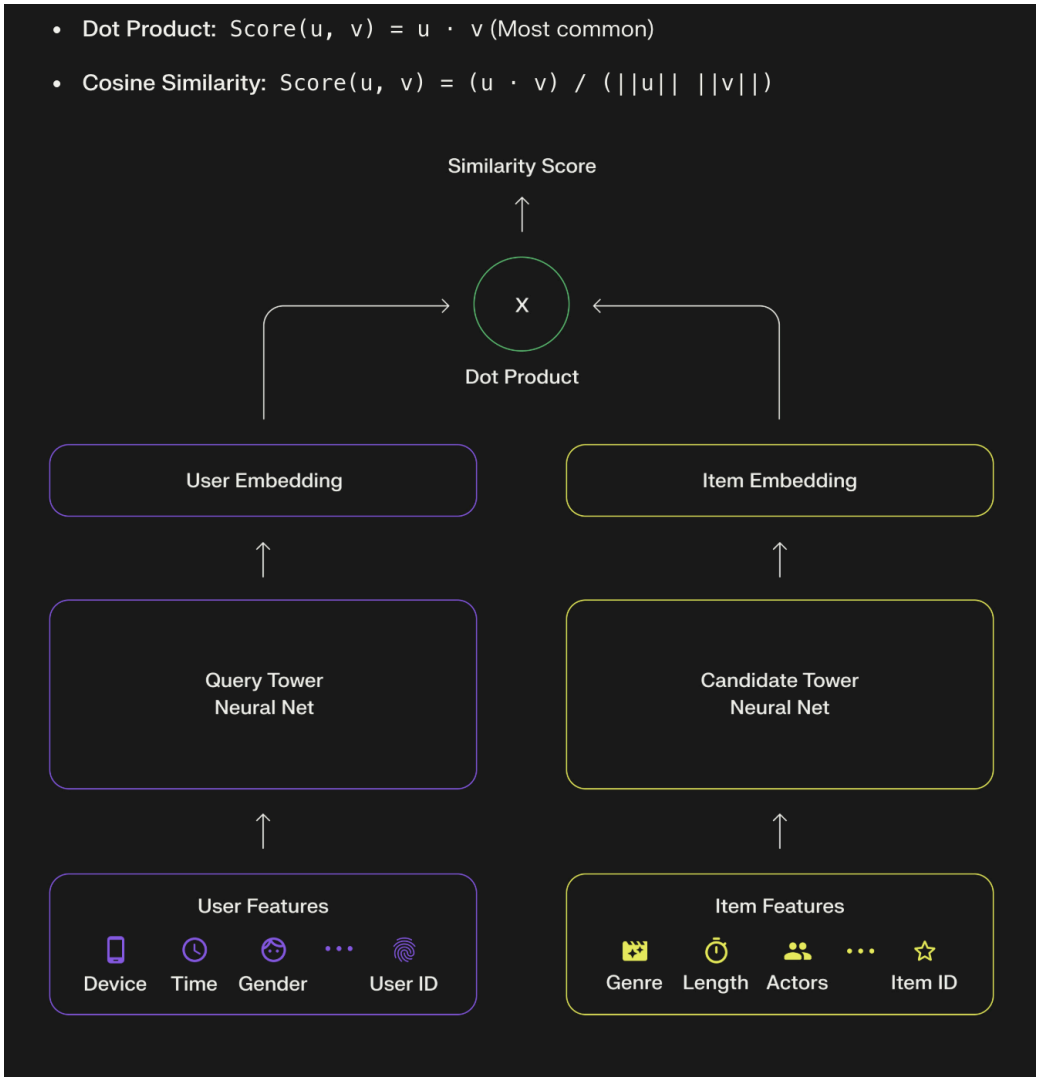
user_id	country	Supported_languages
1001	US	[‘en’,’es’,’hi’]

Post Table:

post_id	post_owner_id	effective_country	Lang
1001	US	IN	En

Algorithm Used: Two-Tower Network

The **Two-Tower** (dual encoder) model is a neural network architecture commonly used in recommendation systems and information retrieval. It gets its name from having two separate neural network "towers": one for the **user** (often called the *query tower*) and one for the **item** (the *candidate tower*). Each tower learns to **encode** its inputs (user features or item features) into a dense **embedding vector** in a shared feature space. The core idea is that if a user and an item are a good match, their embedding vectors will be close or aligned in this space, which we measure by a **similarity score** (usually a dot product).



Why Two-Tower Model?

It enables scalable and flexible recommendations by learning separate embeddings for users and items, allowing fast retrieval via nearest-neighbor search—ideal for large catalogs.
Unlike traditional matrix factorization, it incorporates rich user/item features using neural networks, making it effective for personalized candidate generation in real-time systems.

System Architecture :

User Data → User Tower ┐

├→ Shared Embedding Space → Similarity Scoring → Ranked Results

Item Data → Item Tower ┘

Components :

Component	Description
User Tower	Learns representations from user features (e.g., country, language).
Item Tower	Learns representations from item features (e.g., country, language, tags).
Embedding Space	Shared vector space for user and item embeddings.
Similarity Function	Computes similarity (e.g., dot product or cosine) between user and item vectors.

Two - Tower Network Algorithm Working :

- **User tower**
- Embeddings: `user_id_emb(32)`, `user_country_emb(8)`, `user_lang_emb(8)`
- Concatenate → `user_features`
- MLP: `Dense(32, relu)` → `user_vector = Dense(32, linear)`
- **Item tower**
- Embeddings: `item_id_emb(32)`, `item_country_emb(8)`, `item_lang_emb(8)`
- Concatenate → `item_features`
- MLP: `Dense(32, relu)` → `item_vector = Dense(32, linear)`
- **Similarity:** `Dot(axes=1)` between `user_vector` and `item_vector`
- **Output activation:** `sigmoid` → probability of interaction

Loss & Optimisation -

- Loss: binary_crossentropy
- Optimizer: adam
- Metrics tracked: accuracy

Hyperparameters -

- Embedding dims: 32 (IDs), 8 (country/lang)
- Hidden units per tower: 32
- Batch size: 256
- Epochs: 5
- Validation split: 0.1

Tech Stack -

- Language: Python 3
- Libraries: TensorFlow/Keras, Pandas, NumPy, (standard Python `random`, `ast`)
- Training device: CPU/GPU (Keras-compatible)

Output of Model -

user_id	recommended_post_id	score	rank
10114	2655	0.624968	1
10114	2701	0.585291	2
10114	713	0.578722	3
10000000942	2655	0.741727	1
10000000942	2701	0.703937	2
10000000942	713	0.697708	3
10000001189	2655	0.721780	1
10000001189	2701	0.683907	2

Accuracy of Model -

```
Final training accuracy: 90.01557230949402 %
Final validation accuracy: 6.69826865196228 %
```

Further Improvements -

Need to train on also negative sampling to improve validation accuracy. So, we can do train on likes = 0.

End of Document