https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset?resource=download

---

Technical Documentation

Project: Product Embedding-based Recommendation System

Overview

This document describes the end-to-end architecture, data preparation, embedding neural network design, training, and serving strategy of a product recommendation system based on embedding techniques. The embeddings are learned from sequences of product views by users, leveraging the assumption that products viewed in similar contexts have similar characteristics.

---

1. Dataset

Source:

- **Retail Rocket eCommerce Dataset (Kaggle)**
  - events.csv: timestamped product view data
  - Columns: timestamp, visitorid, event, itemid

Raw Data Sample:

timestamp	visitorid	event	itemid
2015-06-02 05:02:12	257597	view	355908
2015-06-02 05:50:14	992329	view	248676
...	...	...	...

---

2. Data Preparation Pipeline

a. Sessionization:

- Group sequential product views into user sessions based on inactivity (e.g., 30 minutes).
- **Output:** Sessions as ordered product-view sequences:

SessionID 123 → [355908, 248676, 318965]

b. Generating Training Pairs (Positive samples):

- For every session, generate pairs of viewed products with a window size = 1:

[(248676, 355908), (318965, 248676), ...]

- Here, (target, context) implies context is viewed right before or after target.

c. Negative Sampling:

- For each positive pair (target, context), randomly sample products that did not appear with the target in that session as negative samples.
- **Output:** Final labeled training set:

[(248676, 355908, 1), (248676, 111111, 0), (318965, 248676, 1), (318965, 222222, 0)]

d. Mapping item_id to Integer Indices:

- Embedding layers require dense integer IDs:

item2idx = {248676:0, 318965:1, 355908:2, ...}

idx2item = {0:248676, 1:318965, 2:355908, ...}

---

3. Neural Network Architecture

a. Model Inputs:

- Two input IDs (target, context), both integers representing products.

b. Embedding Layers:

- Separate embedding layers for target and context:

target_embedding ∈ $\mathbb{R}$^(N_items×d)

context_embedding ∈ $\mathbb{R}$^(N_items×d)

- N_items = Total unique products.
- d = Embedding dimension (e.g., 50).

c. Forward Computation:

For single training sample (t, c) pair:

v_t = target_embedding[t]   ∈ ℝ^(1×d)

v_c = context_embedding[c] ∈ ℝ^(1×d)

cos_sim = cosine_similarity(v_t, v_c) # normalized dot-product

dense_output = sigmoid(w · cos_sim + b)

- Output probability (dense_output) predicts whether the given pair (t, c) is legitimate (viewed together).

Architecture Diagram:

```
  [target_id]            [context_id]
      |                     |
      ▼                     ▼
[target_embedding]      [context_embedding]
      |                     |
      ▼                     ▼
    v_t                   v_c
      \ cosine similarity (dot) /
              |
              ▼
          Dense Layer (sigmoid)
              |
              ▼
          Output (ŷ ∈ [0,1])
```

---

4. Training Procedure (Detailed)

a. Loss Function:

- Binary Cross Entropy (BCE):

L(y,y^)=−(ylog(y^)+(1−y)log(1−y^))$L(y, \hat{y}) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$

b. Optimization:

- Optimizer: Adam
- Batch training with mini-batches (e.g., 512 pairs/batch).

c. Parameter Updates:

- Only the rows of embeddings corresponding to the target and context product indices in the current batch are updated.

W_tgt[t] ← W_tgt[t] – η ∇(W_tgt[t])

W_ctx[c] ← W_ctx[c] – η ∇(W_ctx[c])

d. Final Trained Embedding Matrix:

- After training, embeddings are extracted and normalized to create product vectors for serving recommendations.

---

5. Serving / Deployment Architecture

a. Preparation (Offline):

- Embedding matrix E is row-wise L2-normalized for fast cosine similarity computations:

E_normalized[i] = E[i] / ||E[i]||

- Embeddings stored and indexed using Approximate Nearest Neighbor (ANN) search (e.g., FAISS, Annoy).

b. Online Recommendation Workflow:

User views product X:

```
  └─> Fetch embedding vector V_x from E_normalized.
    └─> Perform ANN lookup to find top-N nearest vectors.
      └─> Map nearest vectors' indices back to real product IDs.
        └─> Business filtering (stock, promotions, etc.).
          └─> Return recommendations to frontend.
```

---

6. Evaluation & Validation:

- **Quantitative:**
  - Classification accuracy, precision-recall for the binary prediction task.
  - Offline metrics: Hit Rate, NDCG, MRR.
- **Qualitative:**
  - Inspect embedding quality by visualizing using T-SNE or UMAP.

---

7. Low-Level Technical Specifications:

Frameworks & Libraries:

- Python, Pandas, NumPy
- TensorFlow / Keras / PyTorch for neural networks
- FAISS/Annoy for ANN search

Model Serving:

- Embeddings loaded into memory or ANN indices.
- RESTful API (FastAPI, Flask, or similar) for serving requests.

---

8. Limitations & Extensions:

Limitations:

- Cold-start problem for new products without embedding vectors.
- Does not inherently capture user personalization.

Possible Extensions:

- Integrate user embeddings to personalize recommendations.
- Combine metadata embeddings (images, text) with product embeddings.
- Periodically retrain embeddings to adapt to product catalog changes.

---

9. Summary of the Flow:

Raw events → Sessionization → Positive/Negative Sampling → Integer Mapping →

Neural Network Training → Extract Embeddings → Normalize → ANN Indexing → Serve Recommendations

---

Conclusion

This document thoroughly captures the data-flow, computations, technical aspects, and low-level architecture necessary to implement a scalable, efficient embedding-based recommender system. This architecture ensures recommendations leverage implicit user interaction signals, significantly enhancing user experience and product discovery.
