```
!pip install implicit

Collecting implicit
  Downloading implicit-0.7.2.tar.gz (70 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/70.3 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 70.3/70.3 kB 3.0 MB/s eta
0:00:00
ents to build wheel ... etadata (pyproject.toml) ... ent already
satisfied: numpy>=1.17.0 in /usr/local/lib/python3.12/dist-packages
(from implicit) (2.0.2)
Requirement already satisfied: scipy>=0.16 in
/usr/local/lib/python3.12/dist-packages (from implicit) (1.16.3)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.12/dist-packages (from implicit) (4.67.1)
Requirement already satisfied: threadpoolctl in
/usr/local/lib/python3.12/dist-packages (from implicit) (3.6.0)
Building wheels for collected packages: implicit
  Building wheel for implicit (pyproject.toml) ... plicit:
filename=implicit-0.7.2-cp312-cp312-linux_x86_64.whl size=10798000
sha256=69fb413bd998339a7a9af2351ec0153c981af19064b0a065c05461d933b40f7
5
  Stored in directory:
/root/.cache/pip/wheels/b2/00/4f/9ff8af07a0a53ac6007ea5d739da19cfe147a
2df542b6899f8
Successfully built implicit
Installing collected packages: implicit
Successfully installed implicit-0.7.2

import os
os.environ["OPENBLAS_NUM_THREADS"] = "1"

import math
import numpy as np
from scipy.sparse import csr_matrix
from collections import defaultdict
from tqdm import tqdm

import implicit
from implicit.nearest_neighbours import bm25_weight

# ============================================
# CONFIG
# ============================================
INPUT_PATH = "train.txt"
OUTPUT_PATH = "top20_recommendations_bm25_als_pop.txt"
TOP_K = 20

# HYPERPARAMETERS USED
ALPHA = 30.0       # confidence scaling for implicit feedback
FACTORS = 256      # latent dimensions
```

```
REG = 0.02         # regularization
ITERS = 40         # ALS iterations
POP_LAMBDA = 0.15  # weight for popularity in final score

# ============================================
# 1. LOAD DATA (user_id item1 item2 ...) WITH ORDER
# ============================================
# We keep:
# - user_seq: list of items in order (for splitting)
# - user_interactions_full: set of all items per user (for final
model)
user_seq = defaultdict(list)
user_interactions_full = defaultdict(set)
item_pop_counts_full = defaultdict(int)
all_users = set()
all_items = set()

with open(INPUT_PATH, "r") as f:
    for line in f:
        parts = line.strip().split()
        if len(parts) <= 1:
            continue

        u = int(parts[0])
        items = list(map(int, parts[1:]))

        all_users.add(u)
        for it in items:
            user_seq[u].append(it)
            user_interactions_full[u].add(it)
            all_items.add(it)
            item_pop_counts_full[it] += 1

print(f"Total raw users: {len(all_users)}, total items:
{len(all_items)}")

# Global item index over ALL items
item2idx_full = {it: i for i, it in enumerate(sorted(all_items))}
idx2item_full = {i: it for it, i in item2idx_full.items()}
num_items_full = len(item2idx_full)

# ============================================
# 2. TRAIN / VAL / TEST SPLIT PER USER (for evaluation)
# ============================================
train_u2i = {}
val_u2i = {}
test_u2i = {}

for u, items in user_seq.items():
    # remove exact duplicates but KEEP ORDER
```

```
    seen = set()
    unique_items = []
    for it in items:
        if it not in seen:
            seen.add(it)
            unique_items.append(it)

    if len(unique_items) < 5:
        # too few interactions for meaningful splitting -> skip from
eval
        continue

    n = len(unique_items)
    train_cut = int(0.7 * n)
    val_cut = int(0.8 * n)  # 70/10/20 split

    train_items = unique_items[:train_cut]
    val_items = unique_items[train_cut:val_cut]
    test_items = unique_items[val_cut:]

    train_u2i[u] = set(train_items)
    val_u2i[u] = set(val_items)
    test_u2i[u] = set(test_items)

eval_users = sorted(train_u2i.keys())
print(f"Users used for eval (>=5 interactions): {len(eval_users)}")

# Popularity for EVAL (train-only, to avoid leakage)
item_pop_eval = np.zeros(num_items_full, dtype=np.float32)
for u, items in train_u2i.items():
    for it in items:
        item_pop_eval[item2idx_full[it]] += 1.0

# ============================================
# 3. BUILD USER–ITEM TRAIN MATRIX (only TRAIN interactions)
# ============================================
user2idx_eval = {u: i for i, u in enumerate(eval_users)}
idx2user_eval = {i: u for u, i in user2idx_eval.items()}
num_users_eval = len(user2idx_eval)

rows, cols, data = [], [], []
for u, items in train_u2i.items():
    u_idx = user2idx_eval[u]
    for it in items:
        i_idx = item2idx_full[it]
        rows.append(u_idx)
        cols.append(i_idx)
        data.append(1.0)

X_train = csr_matrix((data, (rows, cols)), shape=(num_users_eval,
```

```
num_items_full))

# ============================================
# 4. APPLY BM25 + IMPLICIT CONFIDENCE ON TRAIN
# ============================================
X_bm25 = bm25_weight(X_train).tocsr()
X_conf = (X_bm25 * ALPHA).astype("double").tocsr()

# ============================================
# 5. TRAIN ALS MODEL ON TRAIN MATRIX (for evaluation)
# ============================================
model_eval = implicit.als.AlternatingLeastSquares(
    factors=FACTORS,
    regularization=REG,
    iterations=ITERS,
    random_state=42,
)

print(
    f"\n[Eval model] Training ALS (BM25 + implicit confidence) with "
    f"factors={FACTORS}, reg={REG}, iters={ITERS}, alpha={ALPHA}..."
)
model_eval.fit(X_conf)
print("Eval model training complete.")

# ============================================
# 6. METRIC FUNCTIONS
# ============================================
def recall_at_k(pred, truth, k=TOP_K):
    if not truth:
        return 0.0
    pred_topk = pred[:k]
    return len(set(pred_topk) & truth) / len(truth)

def precision_at_k(pred, truth, k=TOP_K):
    if k == 0:
        return 0.0
    pred_topk = pred[:k]
    return len(set(pred_topk) & truth) / k

def ap_at_k(pred, truth, k=TOP_K):
    if not truth:
        return 0.0
    score, hits = 0.0, 0
    for i, p in enumerate(pred[:k], start=1):
        if p in truth:
            hits += 1
            score += hits / i
    return score / min(len(truth), k)
```

```python
def ndcg_at_k(pred, truth, k=TOP_K):
    if not truth:
        return 0.0
    dcg = 0.0
    for i, p in enumerate(pred[:k], start=1):
        if p in truth:
            dcg += 1.0 / math.log2(i + 1)
    idcg = sum(1.0 / math.log2(i + 1) for i in range(1,
min(len(truth), k) + 1))
    return dcg / idcg if idcg > 0 else 0.0

# ============================================
# 7. EVALUATE ON VALIDATION AND TEST
# ============================================
val_recalls, val_precs, val_maps, val_ndcgs = [], [], [], []
test_recalls, test_precs, test_maps, test_ndcgs = [], [], [], []

print(f"\nEvaluating on validation and test (TOP_K={TOP_K})...")

for u_idx in tqdm(range(num_users_eval)):
    u = idx2user_eval[u_idx]
    train_items = train_u2i[u]
    val_items = val_u2i[u]
    test_items = test_u2i[u]

    seen_indices = {item2idx_full[it] for it in train_items}

    # Ask for more than TOP_K so we can filter & re-rank
    N_raw = TOP_K + len(seen_indices) + 100

    ids, als_scores = model_eval.recommend(
        userid=u_idx,
        user_items=None,
        N=N_raw,
        filter_already_liked_items=False,
        recalculate_user=False,
    )

    ids = np.asarray(ids)
    als_scores = np.asarray(als_scores)

    cand_pop = item_pop_eval[ids]
    pop_scores = np.log1p(cand_pop)
    final_scores = als_scores + POP_LAMBDA * pop_scores

    candidates = []
    for item_idx, score in zip(ids, final_scores):
        if item_idx in seen_indices:
            continue
        candidates.append((item_idx, score))
```

```
    candidates.sort(key=lambda x: x[1], reverse=True)
    topk_indices = [it for (it, _) in candidates[:TOP_K]]
    pred_items = [idx2item_full[i] for i in topk_indices]

    # ---- metrics on VALIDATION ----
    val_truth = val_items
    val_recalls.append(recall_at_k(pred_items, val_truth))
    val_precs.append(precision_at_k(pred_items, val_truth))
    val_maps.append(ap_at_k(pred_items, val_truth))
    val_ndcgs.append(ndcg_at_k(pred_items, val_truth))

    # ---- metrics on TEST ----
    test_truth = test_items
    test_recalls.append(recall_at_k(pred_items, test_truth))
    test_precs.append(precision_at_k(pred_items, test_truth))
    test_maps.append(ap_at_k(pred_items, test_truth))
    test_ndcgs.append(ndcg_at_k(pred_items, test_truth))

print("\n===== VALIDATION METRICS =====")
print(f"Recall@{TOP_K}:    {np.mean(val_recalls):.4f}")
print(f"Precision@{TOP_K}: {np.mean(val_precs):.4f}")
print(f"MAP@{TOP_K}:       {np.mean(val_maps):.4f}")
print(f"NDCG@{TOP_K}:      {np.mean(val_ndcgs):.4f}")

print("\n===== TEST METRICS =====")
print(f"Recall@{TOP_K}:    {np.mean(test_recalls):.4f}")
print(f"Precision@{TOP_K}: {np.mean(test_precs):.4f}")
print(f"MAP@{TOP_K}:       {np.mean(test_maps):.4f}")
print(f"NDCG@{TOP_K}:      {np.mean(test_ndcgs):.4f}")

# ================================================
# 8. TRAIN FINAL MODEL ON ALL DATA & WRITE TOP-20 FILE
# ================================================
print("\nTraining FINAL model on ALL interactions and writing Top-20
file...")

# Popularity from ALL data for final blending
item_pop_full = np.zeros(num_items_full, dtype=np.float32)
for it, cnt in item_pop_counts_full.items():
    item_pop_full[item2idx_full[it]] = cnt

# Full user index over ALL users
all_users_sorted = sorted(all_users)
user2idx_full_users = {u: i for i, u in enumerate(all_users_sorted)}
idx2user_full_users = {i: u for u, i in user2idx_full_users.items()}
num_users_full_users = len(all_users_sorted)

rows, cols, data = [], [], []
for u in all_users_sorted:
```

```
    u_idx = user2idx_full_users[u]
    for it in user_interactions_full[u]:
        i_idx = item2idx_full[it]
        rows.append(u_idx)
        cols.append(i_idx)
        data.append(1.0)

X_full = csr_matrix((data, (rows, cols)), shape=(num_users_full_users,
num_items_full))

X_full_bm25 = bm25_weight(X_full).tocsr()
X_full_conf = (X_full_bm25 * ALPHA).astype("double").tocsr()

model_full = implicit.als.AlternatingLeastSquares(
    factors=FACTORS,
    regularization=REG,
    iterations=ITERS,
    random_state=42,
)

print(
    f"Training FINAL ALS model with factors={FACTORS}, reg={REG}, "
    f"iters={ITERS}, alpha={ALPHA}..."
)
model_full.fit(X_full_conf)
print("Final model training complete.")

print(f"Generating Top-{TOP_K} recommendations for each user into
{OUTPUT_PATH}...")

with open(OUTPUT_PATH, "w") as out:
    for u_idx in tqdm(range(num_users_full_users)):
        u_id = idx2user_full_users[u_idx]
        seen_items = user_interactions_full[u_id]
        seen_indices = {item2idx_full[it] for it in seen_items}

        N_raw = TOP_K + len(seen_indices) + 100

        ids, als_scores = model_full.recommend(
            userid=u_idx,
            user_items=None,
            N=N_raw,
            filter_already_liked_items=False,
            recalculate_user=False,
        )

        ids = np.asarray(ids)
        als_scores = np.asarray(als_scores)

        cand_pop = item_pop_full[ids]
```

```
        pop_scores = np.log1p(cand_pop)
        final_scores = als_scores + POP_LAMBDA * pop_scores

        candidates = []
        for item_idx, score in zip(ids, final_scores):
            if item_idx in seen_indices:
                continue
            candidates.append((item_idx, score))

        candidates.sort(key=lambda x: x[1], reverse=True)
        topk_indices = [it for (it, _) in candidates[:TOP_K]]
        rec_items = [idx2item_full[i] for i in topk_indices]

        line = " ".join([str(u_id)] + [str(it) for it in rec_items])
        out.write(line + "\n")

print("Done. Top-20 recommendations per user saved to:", OUTPUT_PATH)
```

```
Total raw users: 29858, total items: 40981
Users used for eval (>=5 interactions): 29858

[Eval model] Training ALS (BM25 + implicit confidence) with
factors=256, reg=0.02, iters=40, alpha=30.0...

/usr/local/lib/python3.12/dist-packages/implicit/cpu/als.py:95:
RuntimeWarning: OpenBLAS is configured to use 2 threads. It is highly
recommended to disable its internal threadpool by setting the
environment variable 'OPENBLAS_NUM_THREADS=1' or by calling
'threadpoolctl.threadpool_limits(1, "blas")'. Having OpenBLAS use a
threadpool can lead to severe performance issues here.
  check_blas_config()
```


```
{"model_id":"54d8aba80bd94de3985d9b1d3290937c","version_major":2,"vers
ion_minor":0}
```


```
Eval model training complete.

Evaluating on validation and test (TOP_K=20)...

100%|██████████| 29858/29858 [02:34<00:00, 192.90it/s]


===== VALIDATION METRICS =====
Recall@20:    0.1574
Precision@20: 0.0168
MAP@20:        0.0483
NDCG@20:       0.0834

===== TEST METRICS =====
Recall@20:    0.1333
Precision@20: 0.0310
```

```
MAP@20:         0.0450
NDCG@20:        0.0935

Training FINAL model on ALL interactions and writing Top-20 file...
Training FINAL ALS model with factors=256, reg=0.02, iters=40,
alpha=30.0...

{"model_id":"67cc00a193314598a684c5f0c1c0851b","version_major":2,"vers
ion_minor":0}

Final model training complete.
Generating Top-20 recommendations for each user into
top20_recommendations_bm25_als_pop.txt...

100%|██████████| 29858/29858 [02:33<00:00, 194.46it/s]

Done. Top-20 recommendations per user saved to:
top20_recommendations_bm25_als_pop.txt
```