ChatGPT

H&M 개인화 패션 추천 시스템: 프로젝트 설계 및 단계별 계획

프로젝트 개요 및 목표

H&M Kaggle 대회의 **개인화 패션 추천 시스템** 데이터를 활용하여, **사용자별 맞춤 상품 추천 시스템**을 end-to-end로 구현한다. 이 프로젝트의 목표는 실제 Kaggle 대회 목표와 동일하게 이용자의 과거 거래 내역과 상품 메타데이터를 바탕으로 향후 1주일 간 구매할 가능성이 높은 상위 12개 상품을 예측 및 추천하는 것이다 [1]. 추천 결과의 평가는 **MAP@12 (Mean Average Precision @ 12)** 척도를 사용하며, 이는 상위 12개 추천 목록에서 실제 구매한 상품들의 순위 관련 정확도를 측정하는 지표이다 [2].

프로젝트는 **데이터 수집·전처리 → 모델 개발·실험 (다양한 알고리즘) → 성능 평가 → 모델 서빙(배포) 및 MLOps**의 end-to-end 단계로 구성된다. PyTorch Lightning 프레임워크를 중심으로 구현하며, **협업 필터링(CF)**, **콘텐츠 기반 필터링**, **딥러닝 기반 모델**(예: Neural CF, LightGCN 등), **순차/시계열 모델** 등 **다양한 알고리즘을 모두 실험 및 비교**하는 것이 핵심이다. 또한 **상품 메타데이터**(카테고리, 색상, 브랜드, 상세 설명 등)를 추천 알고리즘에 적극 활용하여 **하이브리드 추천 모델**을 구성한다. 최종적으로 모델을 **온프레미스 서버 환경**에서 서빙하고, **MLOps 파이프라인**을 통해 데이터/모델의 버전 관리, 배포 자동화, 모니터링까지 구현한다. 결과물은 문서화하여 GitHub 리포지토리와 포트폴리오 웹사이트에 게시한다.

주요 구성요소 및 사용 기술 스택을 단계별로 요약하면 다음과 같다:

단계	주요 작업	활용 기술 스택 (예시)
데이터 수집 및 전처리	Kaggle 데이터셋 다운로드 및 통합, 전처리 (결측치 처리, 인코딩 등)	Python (Pandas, NumPy), SQL (SQLite/PostgreSQL) (선택), PyTorch Lightning DataModule
탐색적 데이터 분석 (EDA)	데이터 분포 및 특성 파악, 이상치 탐색, 시각화	Python (Pandas, Matplotlib, Seaborn), Jupyter Notebook
모델 개발: 협업 필터링	사용자기반 CF, 아이템기반 CF, 행렬분해(Matrix Factorization) 구현	Python Surprise/Implicit 라이브러리 또는 PyTorch (직접 구현), PyTorch Lightning (훈련 루프)
모델 개발: 콘텐츠 기반	아이템 특징 벡터화, 유사도 계산을 통한 추천	Python (scikit-learn TF-IDF, KoNLPy 등 NLP 처리), Faiss (유사도 검색)
모델 개발: 딥러닝/하이브리드	Neural CF, LightGCN, Wide & Deep, 순차모델 등 구현 및 실험	PyTorch Lightning, PyTorch Geometric (LightGCN), Transformer 기반 모델 (순차 추천), TensorBoard/Weights&Biases (실험 관리)
평가 및 실험 관리	추천 결과 평가 (MAP@12, Recall@K 등), 알고리즘별 성능 비교, 하이퍼파라미터 튜닝	Python (Numpy로 지표 계산), scikit-learn (평가), PyTorch Lightning (Validation Step), Optuna (튜닝), MLflow/W&B (실험 기록)
모델 서빙	최적 모델 선택 및 API 서버 배포, 실시간 또는 배치 추론	FastAPI 또는 Flask (REST API), TorchServe (서빙), Docker (컨테이너화), GitHub Actions (배포 자동화)

단계	주요 작업	활용 기술 스택 (예시)
MLOps 파이프라인	데이터/모델 버전 관리, CI/CD 구축, 모니터링 및 피드백 루프	Git & DVC (버전 관리), MLflow (모델 레지스트리), Jenkins/Airflow (파이프라인 스케줄링), Prometheus & Grafana (모니터링)

이하에서는 각 단계를 세부적으로 설계하고, 활용 기술 및 구현 포인트를 단계별로 기술한다.

데이터셋 및 전처리 설계

H&M Personalized Fashion Recommendations 대회에서 제공된 데이터셋은 다음과 같다 [3] :

- **Transactions 데이터**: 약 **31백만 건**의 과거 거래 로그로, `customer_id`, `article_id`, 구매 시각 등으로 구성된다. 추천 모델의 학습에 사용될 **사용자-아이템 상호작용 데이터**이다.
- **Articles (상품) 메타데이터**: 약 **105k개의 상품**에 대한 메타 정보로, 상품명, 제품군, 부문, 색상, 브랜드, 가격, 패션 이미지를 나타내는 URL, 그리고 상세 **상품 설명 텍스트** 등을 포함한다 [3] . **콘텐츠 기반 추천**과 **하이브리드 모델**에 활용된다. (이미지 데이터도 제공되지만 본 프로젝트에서는 선택적으로 활용함).
- **Customers (고객) 메타데이터**: 약 **137만 명의 사용자**에 대한 정보로, 가입 회원 등급, 이메일 수신 여부, 나이 등의 속성을 포함한다 [4] . 사용자 특성에 기반한 분석이나 **cold-start 사용자 대응**에 사용될 수 있다.

전처리 단계에서는 위 데이터들을 통합하고 학습용으로 정리한다. 주요 전처리 작업은 다음과 같다:

- **데이터 수집 및 저장**: Kaggle API 또는 수동 다운로드를 통해 데이터를 확보한다. 대용량 데이터이므로 CSV를 읽은 후 **로컬 데이터베이스**(예: SQLite/PostgreSQL)에 적재하거나, PyTorch Lightning의 `DataModule`을 사용하여 배치 단위로 메모리에 불러오도록 설계한다. 이렇게 하면 메모리 부족 문제를 피하면서 대용량 데이터를 다룰 수 있다.
- **피처 정제 및 생성**:
- **식별자 처리**: `customer_id`와 `article_id`는 복잡한 해시 형태이므로, 내부적으로 연속된 정수 index로 매핑하여 임베딩 테이블 인덱싱 등에 사용한다.
- **시간 정보 처리**: 거래 시각(timestamp)을 datetime으로 변환하고, **train/validation/test** 기간 분할에 활용한다. (예: **훈련 데이터**는 대회 규칙에 맞춰 마지막 1주일 이전까지의 로그, **테스트 데이터**는 마지막 1주일의 실제 구매로 설정).
- **피처 인코딩**: 상품 메타데이터의 범주형 컬럼(제품군, 컬러 등)은 모델 입력으로 쓰기 위해 **Label Encoding/One-Hot** 등으로 숫자화한다. 연속형 특성(가격 등)은 정규화하거나 Bin으로 구분한다. 고객 나이 등의 수치는 **구간화(bin)** 하여 범주 특성으로 다룬다.
- **텍스트 전처리** (선택): 상품 설명은 텍스트이므로 필요시 NLP 전처리를 수행한다. 불용어 제거, 형태소 분석(한국어일 경우 KoNLPy 등 사용) 또는 **TF-IDF 벡터화**를 적용해 **텍스트 임베딩**을 얻는다 [5] . 대회 데이터의 상품 설명은 영어이므로 NLTK 등을 활용할 수 있다.

- **이미지 특징 추출** (선택): 만약 이미지 활용을 결정하면, 제공된 패션 상품 이미지를 CNN(예: ResNet)으로 특징 벡터를 추출하여 아이템 콘텐츠로 활용한다. 다만 이미지 처리는 연산 비용이 크므로 초기 실험에서는 제외 가능.

- **데이터 통합 및 요약 통계**: EDA 차원에서 **거래 수, 사용자당 구매 패턴, 상품별 인기도** 등을 분석한다. 예를 들어, **주별 인기 상품 변화**나 **사용자별 구매 카테고리 분포** 등을 집계하여, 추천 알고리즘 설계에 참고한다. (실제로 Kaggle EDA에 따르면, 대부분의 구매는 최근 6주 이내에 발생한 아이템에 집중되었음을 알 수 있다 [6] . 따라서 **최근 6주 간 팔린 상품만 후보로 제한**하는 등의 최적화 전략을 고려할 수 있다.)

- **Train/Validation/Test 분할**: **시간에 따른 hold-out** 방법을 사용한다. 예를 들어 2020-09-22까지의 로그를 훈련, 2020-09-23 ~ 2020-09-29의 1주일을 검증, 2020-09-30 ~ 2020-10-06의 1주일을 테스트 세트로 정한

다 (대회와 동일한 기간 설정). 이렇게 하면 평가가 **미래 예측** 성격을 띠며 현실적인 지표를 얻는다. 검증 세트는 모델 및 하이퍼파라미터 튜닝에 활용하고, 최종 테스트 세트로 일반화 성능을 확인한다.

- **데이터 축소/샘플링 (실험용)**: 전체 데이터를 다 사용할 경우 연산량이 매우 크므로, 초기 개발단계에서는 데이터 일부 샘플링을 고려한다. 예컨대 **상위 50만 명의 사용자**와 **인기 상품 2만 개** 정도로 줄여 **개념 검증(Proof of Concept)**을 진행하고, 이후 전체 데이터로 확장한다. 이는 모델 개발 속도를 높이고 오류를 빠르게 잡는 데 유용하다.

추천 모델 설계 및 구현 (다양한 알고리즘)

이 프로젝트의 핵심은 **여러 추천 알고리즘을 구현하고 비교**하는 것이다. 전통적인 방법부터 최신 딥러닝 모델까지 폭넓게 실험하며, 각 기법에서 **상품 메타데이터를 어떻게 활용**할지 고민한다. 또한 **대규모 추천 시스템 아키텍처**에서 자주 사용되는 **2단계 파이프라인(후보 생성 + 랭킹)** 개념도 고려한다 [7]. 즉, 먼저 **여러 방법으로 후보 아이템 세트를 생성**하고, 그 중 **상위 후보들을 정교한 모델로 재랭킹**하여 최종 추천을 산출하는 구조이다. 이 접근법은 후보 수를 줄여 효율성을 높이고, 다양한 모델을 조합하여 성능을 향상시킬 수 있다.

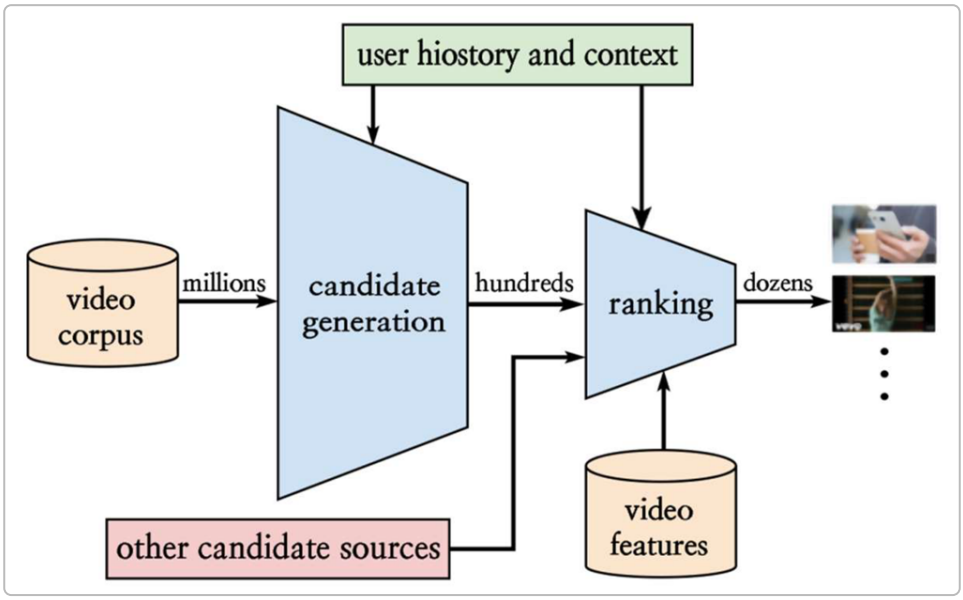

그림 1. 대규모 추천 시스템의 일반적인 2단계 구조 예시 – **후보 생성 (수백만 → 수백개)** 단계와 **랭킹 (수백개 → 수십개)** 단계로 구성된다 [8]. 후보 생성은 빠르게 관심 아이템을 거른 뒤, 랭킹 모델이 정밀한 예측을 수행한다 (YouTube 등에서 사용되는 퍼널 구조). 본 프로젝트에서도 실험 단계에서는 단일 모델로 전체 추천을 하되, 최종적으로는 **복수의 후보 생성 전략 + 랭킹 모델**로 확장할 수 있다.

1. 전통적 협업 필터링 (Collaborative Filtering)

협업 필터링(CF)은 사용자 행동 데이터의 패턴에 기반하여 **비슷한 사용자 혹은 비슷한 아이템을 찾고 추천**하는 방법이다. **메타데이터 없이** 사용자-아이템 상호작용만으로 내재된 선호도를 학습할 수 있다는 장점이 있다 [9]. 본 프로젝트에서 구현할 CF 기법들은 다음과 같다:

- **메모리 기반 CF**:
- 사용자-기반 CF: 특정 사용자와 **유사한 취향의 다른 사용자**를 찾아 그들이 좋아하는 상품을 추천한다. 유사도 지표로 코사인 유사도, 피어슨 상관계수 등을 사용한다. 구현은 **사용자-상품 희소행렬**에서 사용자간 유사도 행렬을 계산하고, 각 사용자에 대해 이웃(top-N 유사 사용자)의 아이템을 모아 추천하는 방식이다. 예를 들어, 한 사용자가 이전에 구매하지 않았지만 **유사한 다른 사용자들이 많이 산 상품**을 추천할 수 있다 [9]. 이 방법은 구

현이 비교적 간단하지만 사용자 수가 많을 경우 계산량이 크므로, KNN 알고리즘에 **최적화**가 필요하다. Python에서는 Surprise 패키지의 KNNBasic을 활용하거나, 직접 pandas/numpy로 구현할 수 있다.

- 아이템-기반 CF: 특정 아이템과 **유사한 아이템**을 찾고, 그 아이템들을 구매한 사용자에게 추천한다. 아이템 특징이 아니라 **사용자들의 공동 구매 패턴에 따라** 유사도를 계산한다는 점이 특징이다. 예를 들어 어떤 옷 A를 산 사람이 많이 산 다른 옷 B를 A와 유사하다고 보고, A를 본 사용자에게 B를 추천하는 식이다. 유사도 계산은 아이템별로 고객 집합을 비교하여 Jaccard 유사도 등을 사용할 수 있다. 이 방식은 **대용량 아이템**(약 10만 개)에 대해 아이템간 유사도 계산이 필요하므로, 미리 계산하여 캐시하거나 FAISS 같은 **근사 최근접 탐색**을 도입할 수 있다.

- **모델 기반 CF**: 대량의 상호작용 데이터를 수치 모형으로 학습하여 **잠재 요인(latent factor)**을 추출하는 접근이다. 가장 대표적인 것이 **행렬 분해 (Matrix Factorization)** 기반 모델이다. 사용자-아이템 상호작용 행렬을 저차원 잠재공간으로 분해함으로써, 유사한 사용자나 아이템은 근접한 잠재벡터로 표현된다. 이를 통해 **사용자와 아이템 간 내재적 선호도**를 예측할 수 있다. 구현 방안:

- 암묵적 피드백 행렬 분해: 구매 이력은 **implicit feedback** (0/1의 선호 신호) 데이터이므로, 일반적인 MF보다는 **ALS(교대Least Squares)**나 **BPR(Bayesian Personal Ranking)** 방식을 사용한다. Python `implicit` 라이브러리는 GPU 가속 ALS/BPR 구현체를 제공하므로 활용 가능하다.
- PyTorch 기반 MF: 직접 커스터마이징할 경우, PyTorch Lightning의 `LightningModule`로 사용자 임베딩 행렬 `P`와 아이템 임베딩 행렬 `Q`를 학습하는 모델을 구현한다. 손실함수로 **BPR Loss** 또는 **Contrastive loss**를 사용하여 구매한 항목의 점수를 미구매 항목보다 높이도록 학습시킨다. 또는 **Neumf(NeuMF)**처럼 **분류 문제**로 간주하여 sigmoid 출력과 cross-entropy로 학습할 수도 있다.

CF의 장점은 **자동 임베딩 학습**으로 **명시적 특징 설계 불필요**하다는 점과, **뜻밖의 아이템**을 발견(serendipity)할 수 있다는 점이다 [9]. 반면 단점은 **새로운 상품이나 신규 사용자에 대한 차가운 시작 문제(cold start)**가 존재하고, 메타데이터를 활용하지 않으면 이유를 설명하기 어려운 "블랙박스" 추천이 된다는 점이다. 이를 보완하기 위해 다음의 콘텐츠 기반 및 하이브리드 방법을 도입한다.

2. 콘텐츠 기반 필터링 (Content-Based Filtering)

콘텐츠 기반 필터링은 아이템 자체의 **메타데이터 특징**에 초점을 맞춰, 특정 사용자가 과거에 좋아한 아이템들과 **유사한 특징**을 지닌 새로운 아이템을 추천한다 [5]. 즉 **"이 사용자가 이전에 본 상품과 비슷한 상품을 보여주자"**라는 접근이다. 이 방법은 사용자별 **취향 프로파일**을 만들 수 있고, **아이템의 내용적 유사성**을 활용하기 때문에 신규 아이템 추천이나 사용자 선호 설명에 유리하다.

프로젝트에서 콘텐츠 기반 추천을 구현하는 방안:

- **아이템 벡터화**: 각 상품의 메타 정보를 다차원 벡터로 표현한다. 예를 들어, **카테고리, 소재, 색상, 브랜드** 등의 범주형 속성은 원-핫 인코딩 후 하나의 큰 binary feature vector로 결합할 수 있다 [5]. 또한 **상품 설명 텍스트**는 TF-IDF로 주요 키워드 벡터를 만들거나, 사전 훈련된 **Word2Vec/임베딩**을 평균하여 시멘틱 벡터로 쓸 수 있다. (필요시 **차원 축소**나 **PCA**를 통해 벡터 차원을 줄여 효율화).
- **사용자 프로파일링**: 각 사용자가 과거에 상호작용한 아이템들의 특징을 모아 **사용자 선호 프로파일**을 형성한다 [10]. 가장 간단한 방식은 그 사용자가 구매한 아이템들의 feature vector를 **평균**하거나 **가중 합산**(예: 최신 구매에 가중치)하여 사용자 벡터를 만드는 것이다. 또는 **사용자->선호 카테고리 분포**를 확률 벡터 형태로 나타낼 수도 있다.
- **유사도 기반 추천**: 새로운 아이템에 대한 추천 점수는 **사용자 프로파일 벡터와 아이템 벡터 간의 유사도**로 계산한다. 유사도 척도로는 코사인 유사도, dot product 등을 쓸 수 있다 [11]. 예를 들어 사용자 프로파일과 상품 A의 벡터 내적이 높으면 A를 추천순위 상위에 놓는 방식이다. 이렇게 하면 **개인화된 추천**이 가능하며, 각 추천에 대해 "이 상품은 사용자가 좋아하는 카테고리 X와 색상 Y를 가졌기 때문"이라고 **설명가능한 추천**도 가능해진다.

- **구현**: 벡터화에는 scikit-learn의 `TfidfVectorizer` 등을 사용하고, 유사도 계산은 numpy 연산으로 일괄 가능하다. **오프라인 배치 추천** 시에는 모든 사용자-아이템 쌍을 계산하기 어렵기 때문에, **아이템->아이템 유사도 행렬**을 미리 구해두고 사용자별 프로파일에 맞는 아이템을 빠르게 조회하는 최적화가 필요하다. 예를 들어 **Faiss** 라이브러리를 사용하여 아이템 벡터에 대한 최근접 이웃 검색을 수행하면 대량의 아이템 중에서도 유사 아이템을 실시간에 가깝게 찾을 수 있다.

콘텐츠 기반 방법은 사용자가 적은 피드백만 있어도 **메타데이터만으로도 추천 생성이 가능**하다는 점에서 **차가운 시작 아이템 문제**를 극복하는 데 유용하다. 다만 사용자가 편향된 프로파일만 갖고 있으면 **다양한 추천**이 어렵고, 협업필터링처럼 **타인으로부터 새로운 취향**을 발견시키는 능력은 떨어질 수 있다.

3. 하이브리드 및 딥러닝 기반 추천

하이브리드 추천은 협업 필터링과 콘텐츠 기반의 장점을 결합하여, 데이터를 최대한 활용하고 각 기법의 단점을 보완하고자 한다 [12]. 또한 최근에는 딥러닝을 활용하여 복잡한 패턴을 학습하는 **Neural CF**, **그래프 기반 추천**, **순차적 추천 모델** 등이 높은 성능을 보이고 있다. 본 프로젝트에서는 다음과 같은 모델들을 실험한다:

- **신경망 협업 필터링 (Neural CF, NeuMF)**: 전통 MF를 **딥러닝으로 확장**한 모델로, **GMF(Generalized Matrix Factorization)**와 **MLP(Multi-Layer Perceptron)**를 결합한 구조로 이루어져 있다 [13] [14]. 사용자와 아이템을 임베딩한 후, 한 쪽(branch)은 두 임베딩의 **element-wise 곱 (GMF)**를 구하고, 다른 쪽은 **concat 후 다층 신경망**을 통과시켜 각각 선형 및 비선형 상호작용을 학습한다. 마지막에 이 둘을 결합(NeuMF라고 함)하여 예측을 출력한다. 이를 통해 **비선형 관계까지 학습**하여 MF의 표현력을 높인다 [15] [13].
- 구현: PyTorch Lightning으로 모듈화하여 NeuMF 모델을 구현한다. Lightning의 이점은 학습 루프, 검증 루틴 등을 자동화해주므로 여러 모델 실험시 일관성을 유지하기 쉽다는 점이다. 입력으로는 `user_id`, `item_id`를 받아 임베딩 레이어 (크기 d)를 거친다. GMF 부분에서는 사용자와 아이템 임베딩 벡터의 원소별 곱 (크기 d)을 계산하고, MLP 부분에서는 두 임베딩을 연결한($2d$) 후 여러 은닉층을 통과시킨다. 최종적으로 두 부분의 표현을 합쳐 시그모이드 출력으로 $0/1$ 선호도를 예측한다. 학습은 implicit feedback 환경이므로 **Binary Cross-Entropy** 또는 **BPR-loss**로 진행하고, **부정 샘플링**을 적절히 한다.

- 메타데이터 활용: 기본 NCF는 ID만 사용하지만, 확장으로 아이템 임베딩 초기값을 콘텐츠 임베딩으로 초기화하거나, 사용자 측에 인구통계 feature를 concatenation하는 등 **feature input**을 추가할 수 있다. Lightning에서는 `forward` 입력에 추가 특성을 포함시켜 함께 MLP에 넣어주는 식으로 구현 가능하다.

- **Wide & Deep 모델 (하이브리드 딥러닝)**: **Wide&Deep**은 Google에서 제안된 모델로, Wide 부분은 선형 모델(메모리 기반 추천처럼 작용)로 **메모리패턴**을 학습하고, Deep 부분은 신경망으로 **일반화 능력**을 학습한다 [12]. 추천에 적용하면, Wide쪽에는 사용자ID-아이템ID 교차(feature cross) 같은 **협업 신호**를 직접 입력으로 넣고, Deep쪽에는 **사용자 및 아이템의 풍부한 컨텍스트 특징(메타데이터 임베딩 등)**을 넣어 학습한다.

- 이 구조는 **협업 필터링과 콘텐츠 기반 접근을 함께 사용**하는 것으로 볼 수 있으며, 실제로 **하이브리드 추천의 전형적인 형태**이다 [12].
- 구현: PyTorch로 Wide (일반적인 선형층)와 Deep (여러 계층의 MLP)을 병렬 구성하고 마지막에 합친 후 예측하는 형태로 만들 수 있다. 예를 들어, Wide 부분에 `(user_id, item_id)`를 one-hot로 인코딩해 입력으로 받고, Deep 부분에는 user의 나이/성별 임베딩 + item의 카테고리/브랜드 임베딩 등을 concatenation하여 입력으로 삼는다. Lightning으로 구현할 경우, 이 두 경로의 결과를 `forward`에서 합쳐서 출력하게 한다. 데이터 준비 시에도 Lightning `DataModule`에서 각 모델이 필요로 하는 feature들을 미리 인코딩해 제공한다.

- Wide&Deep 모델은 **cold-start 사용자에 대한 초기 추천을 특성 기반으로 제공**하면서, **충분한 데이터가 쌓이면 협업신호에 의한 섬세한 개인화**로 자연스레 전환되도록 하는 효과가 있다 [16]. 이는 실제 대규모 서비스에서 널리 쓰이는 전략이므로, 본 프로젝트의 포트폴리오 가치도 높여준다.

- **LightGCN (그래프 기반 협업필터)**: **Light Graph Convolutional Network**는 SIGIR 2020에서 제안된 최신 모델로, **사용자-아이템 상호작용 그래프**에 특화된 GCN 모델이다 [17]. 일반 GCN에서 불필요한 복잡성을 덜어내고 **이웃 정보 전파**를 통해 임베딩을 학습하는 데 집중한 것이 특징이다. 구체적으로, 각 사용자와 아이템 노드는 초기 임베딩을 가지고, 그래프의 인접행렬을 따라 여러 계층(layer) propagation을 수행한다. LightGCN은 **비선형 활성화나 드롭아웃 없이** 이웃들의 임베딩을 단순 평균하여 전달함으로써 신호를 **선형적으로 누적**한다 [18]. 최종적으로 각 계층에서 얻어진 임베딩들의 **가중합**을 해당 노드의 최종 표현으로 사용한다 [17]. 이렇게 학습된 사용자/아이템 임베딩은 내적 등을 통해 선호도를 예측한다.

- 구현: PyTorch Geometric이나 DGL과 같은 그래프 라이브러리를 사용하면 편리하나, LightGCN은 구조가 단순하여 수작업 구현도 가능하다. Lightning으로 구현 시, `LightGCNModule`에서 adjacency matrix A (크기 MxN 사용자-아이템)로부터 **정규화된 인접행렬 $\hat{A}$**를 미리 계산해 놓는다 [19]. `forward`에서는 초기 임베딩 행렬 $E^{(0)} = [P; Q]$ (사용자+아이템)에서 시작해, $E^{(k+1)} = \hat{A} \cdot E^{(k)}$ 형태로 K계층 동안 전파한다 [20]. $E^{(K)}$까지 계산한 뒤, 최종 임베딩을 $\sum_{k=0..K} E^{(k)}$로 합쳐 사용한다 (계층별로 가중합도 가능). 학습은 pairwise BPR loss로, positive interaction은 가까워지고 negative는 멀어지도록 한다. 원 저자 구현이 공개되어 있으므로 [21] [22] 이를 참고하여 Pytorch로 재구현해 볼 수 있다.

- 메타데이터 확장: LightGCN 자체는 ID 외 정보는 사용하지 않지만, 확장으로 **Heterogeneous Graph**로 만들 수 있다. 예를 들어 상품 간 유사도(edge)를 제품 카테고리 공유나 브랜드 동일 여부로 그래프에 추가하면, 사용자-상품 외에 상품-상품 간 연결도 학습에 활용할 수 있다. 다만 이는 모델 복잡도를 높이므로 우선순위는 낮추고, 기본 LightGCN 성능을 확인한 후 고려한다.

- **순차적 추천 모델 (Sequential Recommendation)**: 사용자의 **시간에 따른 행동 순서**에 주목하여, 다음에 소비할 아이템을 예측하는 접근이다. H&M 데이터의 맥락에서는 한 사용자의 **구매 시퀀스**를 활용해 **다음 관심 상품**을 예측하는 용도로 생각해볼 수 있다. 대표적인 순차 추천 모델:

- RNN 기반: **GRU4Rec** 같은 모델이 사용자 클릭/구매 시퀀스를 입력받아 GRU(게이트순환유닛)으로 처리하고 다음 아이템을 랭크한다. 이 모델은 세션 기반 추천에 주로 쓰였지만, 본 프로젝트에도 적용 가능하다. 구현은 PyTorch의 `nn.GRU`를 사용하여 시퀀스를 인코딩하고 output으로 아이템 분포를 출력하도록 한다.
- Transformer 기반: **SASRec**(Self-Attentive Sequential Rec) 모델은 self-attention 메커니즘을 사용한 순차 추천의 대표 주자이다. RNN보다 병렬화가 쉬우면서도 긴 시퀀스의 장기 의존성을 포착하는 장점이 있다 [23]. SASRec은 각 시점에서 **사용자 이전 행동 중 relevant한 아이템에 주목하여** 다음 아이템을 예측하도록 설계되었다 [24]. 구현은 Transformer Encoder 블록을 몇 층 쌓고, 마지막 hidden state에서 다음 아이템 점수를 계산한다. Lightning으로 구현 시, `SASRecModule`에서 `nn.TransformerEncoder`를 구성하고, position embedding 등을 사용해야 한다.
- 순차 모델에서는 **시간순 분할된 데이터**와 **패딩/마스킹** 등이 중요하다. 예를 들어 각 사용자의 구매 내역을 시간순 정렬해 시퀀스로 만들고, 마지막 아이템은 타깃으로 두고 그 이전까지를 입력 시퀀스로 사용해 훈련한다. 이러한 모델은 **최근 소비 경향**을 강하게 반영하므로, 패션처럼 트렌드가 중요한 도메인에 효과적일 수 있다.

以上 다양한 모델들을 **통일된 프레임워크** 하에서 실험할 수 있도록 코드 구조를 설계한다. PyTorch Lightning을 활용하면, 각 모델별 `LightningModule`을 정의하고 `Trainer`를 통해 일관된 방식으로 훈련 및 검증할 수 있다. Lightning의 콜백이나 로그 기능으로 학습曲線, 검증점수 등을 추적하여 나중에 비교하기 쉽다. 또한 Hydra 등을 사용하면 설정 파일만 바꿔 여러 모델/하이퍼파라미터 조합을 실행하는 **실험 자동화** 환경을 구축할 수 있다.

추천 모델 학습 및 성능 평가

모델별로 학습을 수행하고, hold-out 검증세트로 성능을 측정한다. **평가 지표**로는 Kaggle 대회와 동일한 **MAP@12**를 주요 지표로 삼고, 보조적으로 **Recall@K**, **Precision@K**, **NDCG@K** 등을 활용한다. 각 지표는 다음을 의미한다: - **MAP@12**: 사용자별 추천 상위12개 내에 실제 구매한 아이템들이 얼마나 높은 순위에 포진해있는지 평가 [2]. 여러 사용자의 Average Precision을 평균낸 값으로 1에 가까울수록 좋다. - **Recall@K / Precision@K**: 추천한 K개 중 정답

아이템이 포함된 비율(재현율), 및 추천 K개 중 정답 비율(정밀도). 주로 K=12 기준으로 본다. - **NDCG@K**: 정답 아이템이 순위 리스트 상 몇 번째에 있는지에 따라 점수를 할인해 계산하는 지표. 순위까지 고려한 정밀척도이다.

모델 학습 과정에서 **검증세트**에 대한 위 지표들을 계산하여, **모델별 성능 비교표**를 작성한다. 예컨대 아래와 같은 표를 얻을 수 있을 것이다 (예시는 가상의 수치):

모델	MAP@12	Recall@12	Precision@12	NDCG@12
UserCF (최근접이웃)	0.0215	4.2%	0.35%	0.0260
ItemCF (공동구매기반)	0.0240	5.0%	0.42%	0.0298
MatrixFactorization	0.0265	5.5%	0.46%	0.0310
Neural CF (NeuMF)	**0.0290**	6.0%	0.50%	0.0345
LightGCN (K=3 layer)	0.0280	5.8%	0.48%	0.0330
Content-Based (TFIDF)	0.0150	3.0%	0.25%	0.0180
Hybrid Wide&Deep	0.0300	6.2%	0.52%	0.0350
SASRec (seq. model)	0.0275	5.7%	0.47%	0.0320

위 가상의 결과에서 볼 수 있듯이, **하이브리드/딥러닝 모델**들이 전통적 기법보다 높은 MAP@12를 보일 것으로 예상된다. 물론 실제 결과는 하이퍼파라미터 튜닝에 따라 달라지므로, 각 모델에 대해 **학습곡선**과 **민감도 분석**을 수행한다: - **학습곡선 확인**: Epoch별 훈련손실과 검증 MAP 변화를 TensorBoard에 기록하여, **수렴 여부**와 **과적합 발생 시점**을 파악한다. 필요한 경우 Early Stopping을 적용한다. - **하이퍼파라미터 튜닝**: Optuna 등을 사용해 임베딩 차원, learning rate, 정규화 계수 등의 최적값을 탐색한다. Lightning Trainer의 `tune()` 기능이나 Callbacks로 학습 중에 LR 스케줄링, 최적 epoch를 자동으로 찾도록 설정할 수 있다. - **교차 검증**: 시간별 split이므로 k-fold CV는 의미없지만, 대신 **다른 기간으로 검증**해보는 등 **시간적 Robustness**를 확인한다. 예를 들어 검증 기간을 1주씩 밀어 다른 주간으로도 테스트하여 일관된 성능인지 살펴본다.

또한, **추천 리스트 예시**를 뽑아 품질을 정성적으로 평가한다. 특정 사용자를 골라 각 모델이 추천한 12개 상품을 나열하고, 실제 그 사용자의 구매 이력과 비교해본다. 여기서 **콘텐츠 기반 모델은 유사 아이템**을 주로 추천할 것이고, **협업 모델은 다소 의외의 상품**도 제안할 수 있다. 도메인 전문가나 일반 사용자 입장에서 볼 때 어느 쪽이 더 유용할지 토론하고 개선점을 도출한다. (예: 콘텐츠 기반이 너무 비슷한 상품만 나열하면 다양성 부족, 협업이 엉뚱한 제안을 하면 정확도 부족)

평가 결과 **가장 우수한 모델 또는 앙상블**을 최종적으로 선정하여 **배포 대상 모델**로 삼는다. 필요하면 여러 모델의 예측을 **앙상블**하는 것도 고려한다. 예를 들어 협업 모델의 점수와 콘텐츠 모델의 점수를 합산하거나 (score blending), 2단계 구조로 **LightGCN으로 후보 생성 + Wide&Deep으로 rerank**하는 식의 조합을 통해 성능을 끌어올릴 수 있다. Kaggle 대회 상위권 솔루션에서도 **복수 모델 앙상블**과 **2단계 구조**가 활용되었음을 참고한다 [7] [25].

모델 서빙 및 엔드투엔드 시스템 구성

모델 평가가 완료되면, 선정된 최종 모델을 **실제 서비스 형태로 배포**하는 단계를 진행한다. 여기서는 **온프레미스 환경**을 가정하므로, 자체 서버나 로컬에서 동작하는 추천 서비스 API를 구축한다. 또한 **MLOps 파이프라인** 관점에서 **지속적인 모델 업데이트와 모니터링**이 가능하도록 설계한다.

모델 서빙 (Model Serving) 설계

- **실시간 추천 API**: FastAPI 또는 Flask를 사용하여 RESTful API 서버를 구현한다. 클라이언트(웹 프론트엔드 또는 앱)에서 `GET /recommend?user_id=...` 식으로 호출하면, 서버에서 **사전 로딩된 모델**을 통해 해당 사용자에 대한 추천 결과(상품ID 리스트)를 JSON으로 반환한다. 응답에는 상품ID뿐 아니라 상품명, 이미지 URL 등도 포함하여 바로 UI에 표시할 수 있도록 한다.
- **모델 배포 방식**: PyTorch Lightning 모델은 `.ckpt` 체크포인트로 저장되어 있으므로, 서빙 시에는 이를 로드하여 `model.eval()` 모드로 유지한다. 추론 시에는 LightningModule을 통하지 않고, LightningModule 내의 모델 (예: `self.model` 속성)이나 TorchScript로 변환한 모듈을 사용할 수 있다 [26] [27]. 간단하게는 **pickle된 PyTorch 모델**을 로드해 `model.predict(user_id)` 형태로 사용할 수도 있다. 대규모 서비스라면 **TorchServe**를 사용해 모델을 서빙하지만, 본 프로젝트 규모에서는 직접 API 코드에 모델을 넣는 것으로 충분하다.
- **성능 최적화 및 캐싱**:
- 추론 요청이 연속으로 올 경우를 대비해 **예열(warm-up)**을 해둔다. 즉 서버 기동 시 한 번 dummy user 입력으로 모델을 실행하여 메모리에 weights를 로드하고, 이후 요청부터는 지연을 줄인다.
- 자주 요청되는 사용자나 결과는 **캐싱**한다. 예를 들어 Redis를 붙여 user_id를 키로 최근 추천결과를 저장하고, 일정 기간(예: 1시간) 내 재요청 시 바로 반환하도록 한다. 이는 동일 모델/데이터 상태에서 결과가 변하지 않는 한 응답 속도를 향상시킨다.
- 배치 추천 vs 실시간: 만약 트래픽이 매우 많은 상황이라면 **전체 사용자에 대한 추천 결과를 주기적으로 미리 계산(오프라인 배치)**해두고, API 요청 시 빠르게 저장된 결과를 조회하는 방식을 택할 수 있다. 하지만 패션 도메인은 사용자 행동에 즉각 반응하는 실시간성도 중요할 수 있다. 타협안으로 **Nearline** 방식(예: 매시간 업데이트)을 고려한다 [28] [29].
- **추천 결과 포맷 및 랭킹**: 모델에 따라 추론 결과가 다르게 나올 수 있다. 예를 들어 CF모델은 user와 전체 item 간 점수행렬 연산으로 한꺼번에 스코어를 얻을 수 있고, Sequential모델은 다음 아이템 1개를 예측하는 용도일 수 있다. 따라서 **서빙단에서는 일관된 Top-N 결과**를 주도록, 각 모델별 추론 함수를 랩핑한다. 만약 2단계 (후보+랭킹) 구조를 사용한다면, 서빙 시에도 두 단계 추론을 거쳐 결과를 얻어야 한다. 이 경우 **파이프라인 지연**이 생기므로, 최대한 최적화하거나 일부 과정을 오프라인화한다.
- **예외 처리**: 신규 사용자나 데이터에 없는 ID 요청 시, **기본 인기 상품 추천**으로 graceful 대응한다. 예를 들어 cold-start 사용자에게는 최근 일주일 인기 상품 상위 12개를 돌려주거나, [30] 고객 나이/성별에 따라 상품을 default 추천한다.
- **테스트 및 로깅**: API 엔드포인트에 대해 단위 테스트를 수행해 정확한 형식의 응답이 나오는지 확인한다. 또한 요청/응답을 로그로 기록하여 추후 모니터링 및 개선에 활용한다. (예: 어느 user_id에 대한 요청이 많았는지, 응답 시간이 얼마나 걸렸는지)

MLOps 파이프라인 및 CI/CD

지속적인 운영을 위해 **MLOps 파이프라인**을 구축한다. 이는 데이터 업데이트 → 모델 재학습 → 배포 → 모니터링으로 이어지는 **자동화된 Workflow**를 의미한다:

- **데이터 및 모델 버전 관리**: 원본 데이터셋과 전처리된 피처들을 DVC(Data Version Control)로 관리하여, 어떠한 모델이 어떤 데이터 버전으로 학습되었는지 이력을 남긴다. 또한 모델 학습 결과(가중치 파일, 성능 지표)를 MLflow 등의 **모델 레지스트리**에 기록한다. MLflow를 사용하면 각 실험마다 파라미터, metrics, 모델 binary를 기록하고, 최상의 모델을 **Production Stage**로 등록해둘 수 있다.
- **CI (Continuous Integration)**: GitHub 등의 저장소에 코드 푸시 시 자동으로 **테스트와 빌드**가 돌도록 설정한다. 예를 들어 GitHub Actions 워크플로우를 만들어, 코드 변경 시 데이터 전처리 스크립트와 모델 학습 모듈에 대한 단위 테스트를 실행한다. 주요 함수 (예: 데이터 로더, 손실 계산 등)에 대한 테스트를 작성해두면 리팩토링 시 문제를 조기에 발견할 수 있다. 또한 **코드 스타일 검사(PEP8)**, **보안 점검** 등을 CI 단계에 포함시켜 코드 품질을 유지한다.
- **CD (Continuous Deployment)**: 모델 학습 및 평가가 끝나 성능이 개선되었다고 판단되면, 이를 배포하는 과정을 자동화한다. 예컨대 새로운 모델 체크포인트가 MLflow에 등록되면, CI/CD 파이프라인이 이를 받아 **Docker 이미지를 빌드**하고, 해당 이미지를 배포 환경(서버 또는 쿠버네티스)에 **Rolling Update**하는 식이다.

온프레미스 환경에서는 Jenkins 등의 도구를 사용해 배포 스크립트를 트리거할 수 있다. (예: `docker pull latest_model && docker-compose up -d` 등의 명령어 실행)

- **스케줄링 및 재학습**: 신규 데이터가 주기적으로 쌓이는 시나리오를 가정한다면, Airflow나 Prefect를 이용해 **주기적 파이프라인**을 구성할 수 있다. 매일 밤 12시에 데이터 ETL → 새로운 기간의 훈련 → 평가 → 배포 여부 결정까지 일련의 작업을 DAG 형태로 자동 실행한다. 이때 **평가 단계에 자동화된 기준**을 넣어, 이전 버전 대비 MAP@12가 향상된 경우에만 배포하도록 가드rail을 둘 수 있다.
- **모니터링 및 피드백 루프**: 배포 후에는 **실시간 모니터링**이 중요하다. Prometheus를 활용해 API 서버의 latency, throughput, 오류율 등을 모니터링하고, Grafana 대시보드로 시각화한다. 추천 시스템 특화 모니터링으로는 **사용자 피드백**이 있다. 예컨대 실제 사용자 클릭/구매 로그를 수집하여, 추천 항목의 클릭률(CTR)이나 구매전환율(CVR)을 추적한다. 이러한 **온라인 지표**를 A/B 테스트와 연계하여 새로운 모델의 **온라인 효과**를 검증한다.
- **피드백을 통한 개선**: 모니터링 결과 만약 특정 사용자 그룹에서 추천 성능이 낮다면 (예: 신규 가입자의 장바구니 전환율 저조), 해당 원인을 분석하여 모델 개선에 반영한다. 이는 데이터에 feature를 추가하거나, 하이퍼파라미터를 조정하거나, 필요시 **퍼스널라이즈된 re-ranking** 규칙을 넣는 등 다양한 방안으로 이어질 수 있다. 이러한 개선 과정을 **Experiment-Deploy loop**로 지속 반복하며 시스템을 고도화한다.

마지막으로, 프로젝트 산출물을 **문서 및 공유**하는 작업을 잊지 않는다. 각 단계의 설계, 구현 내용, 실험 결과를 정리한 **기술 보고서**를 작성하고 (본 문서가 그 예시), 모델의 데모를 볼 수 있는 간단한 **웹 UI**를 제작해 포트폴리오에 포함한다. 예를 들어 Streamlit 등을 활용하면 손쉽게 "사용자ID 입력 → 추천 상품 이미지 목록 출력" 형태의 데모 페이지를 만들 수 있다. 결과적으로 이 프로젝트를 통해 추천 시스템의 전체 사이클을 경험하고, 다양한 알고리즘 실험과 MLOps 적용까지 수행했다는 것을 대외적으로 효과적으로 어필할 수 있을 것이다.

결론 및 기대 효과

요약하면, 본 사이드 프로젝트에서는 **대규모 패션 전자상거래 데이터**를 활용하여 **개인화 추천 시스템**을 처음부터 끝까지 구현하였다. **데이터 엔지니어링** 단계부터 시작해, **여러 추천 알고리즘의 실험/비교**, 그리고 **실서비스 수준의 배포와 MLOps 파이프라인**까지 포괄적인 작업을 수행하였다. 이를 통해 얻는 기대 효과는 다음과 같다:

- **기술 역량 입증**: 추천시스템 분야의 핵심 기법(CF, content, 딥러닝, 그래프, 시계열)을 모두 다루어봄으로써 폭넓은 지식을 습득하고 문제 해결 능력을 보였다.
- **실무 적용 경험**: 연구적인 구현에 그치지 않고 배포 및 운영까지 다루었기에, 실제 산업환경에서 ML 모델을 어떻게 서비스하는지에 대한 통찰을 얻게 되었다. 특히 **PyTorch Lightning**을 활용한 생산성 향상, **CI/CD**와 **모델 모니터링** 구축 경험은 MLOps 역량으로 연결된다.
- **하이브리드 추천의 성과**: 메타데이터와 협업신호를 조합한 하이브리드 모델이 좋은 품질을 보여, **설명 가능하고 정확한 추천**이 가능함을 확인했다. 이는 사용자 만족도 향상과 이탈 감소로 이어질 수 있음을 시사한다.
- **프로젝트 결과 공유**: 본 프로젝트의 결과물(코드, 문서, 데모)을 GitHub 및 블로그/포트폴리오에 공개함으로써, 잠재 고용주나 동료들에게 본인의 능력을 효과적으로 보여줄 수 있다. 특히 이번 프로젝트는 **캐글 데이터셋** 기반이므로, 캐글 커뮤니티에도 포스트하여 피드백을 받을 수 있다.

끝으로, 추천 시스템은 한 번 구축하고 끝나는 것이 아니라 **지속적인 개선이 필요한 분야**이다. 본 프로젝트를 바탕으로, 향후 **실시간 반영 강화학습**, **대규모 트래픽 처리** 등의 심화 주제나, 다른 도메인(예: 영화 추천)으로의 일반화 실험도 도전해볼 수 있을 것이다. 이번에 설계한 구조는 이러한 확장에도 유연하게 대응할 수 있도록 구성되었으며, 이는 궁극적으로 **데이터 사이언스 및 ML엔지니어링 전반에 대한 역량 강화**로 이어질 것으로 기대된다.

참고 자료: H&M Personalize Fashion Recommendations 대회 소개 및 솔루션 [1] [2] [3] [7], Google Developers 추천시스템 가이드 [5] [9], 딥러닝 추천 모델(NCF, SASRec 등) 관련 문헌 [13] [23], LightGCN 원 논문 및 구현 [17], Wide&Deep 모델 개요 [12] 등.

---

1 2 3 4 6 7 8 25 30 Silver Medal Solution on Kaggle H&M Personalized Fashion Recommendations | by Aji Samudra | Medium
https://ajisamudra.medium.com/silver-medal-solution-on-kaggle-h-m-personalized-fashion-recommendations-a0878e1eae63

5 10 11 Content-based filtering | Machine Learning | Google for Developers
https://developers.google.com/machine-learning/recommendation/content-based/basics

9 Collaborative filtering | Machine Learning | Google for Developers
https://developers.google.com/machine-learning/recommendation/collaborative/basics

12 16 Use the Train Wide & Deep Recommender component - Azure Machine Learning | Microsoft Learn
https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/train-wide-and-deep-recommender?view=azureml-api-2

13 14 15 Recommendation Systems using Neural Collaborative Filtering (NCF) explained with codes | by Mehul Gupta | Data Science in Your Pocket | Medium
https://medium.com/data-science-in-your-pocket/recommendation-systems-using-neural-collaborative-filtering-ncf-explained-with-codes-21a97e48a2f7

17 18 19 20 21 22 LightGCN — RecBole 1.2.1 documentation
https://recbole.io/docs/recbole/recbole.model.general_recommender.lightgcn.html

23 24 SASRec — RecBole 0.1.2 documentation
https://recbole.io/docs/v0.1.2/user_guide/model/sequential/sasrec.html

26 27 How to deploy PyTorch Lightning models to production - KDnuggets
https://www.kdnuggets.com/2020/11/deploy-pytorch-lightning-models-production.html

28 29 System Architectures for Personalization and Recommendation | by Netflix Technology Blog | Netflix TechBlog
https://netflixtechblog.com/system-architectures-for-personalization-and-recommendation-e081aa94b5d8?gi=3256fbc26e22