# **Lab Session: Building a Recommender System with LightFM**

## **Overview**

In this lab session, we will go through the entire pipeline of building a recommender system. We will use the [H&M dataset released in a Kaggle competition](https://www.kaggle.com/c/h-and-m-personalized-fashion-recommendations) and the [LightFM library](https://making.lyst.com/lightfm/docs/home.html). The session will cover data analysis, data sampling, model training, hyperparameter tuning, evaluation, and hybrid recommendation incorporating item features.

## **Dataset**

Download the H&M dataset from the Kaggle competition page, or use the data availablein the gdrive folder.

### **Files needed:**

* transactions\_train.csv
* articles.csv
* customers.csv

## **Introduction to LightFM**

### **What is LightFM?**

LightFM is a Python library designed for building and evaluating recommender systems. It is particularly well-suited for handling hybrid recommendation scenarios that combine collaborative filtering with content-based methods. LightFM is known for its flexibility, allowing you to incorporate user and item metadata into the recommendation process, which can significantly improve the accuracy of your recommendations.

### **Key Features of LightFM**

1. **Flexible Hybrid Models**: LightFM allows for the combination of collaborative and content-based filtering by integrating item and user features.
2. **Different Loss Functions**: LightFM supports several loss functions for training models, including:
   * **WARP (Weighted Approximate-Rank Pairwise)**: Optimizes for ranking quality, suitable for implicit feedback data.
   * **BPR (Bayesian Personalized Ranking)**: Optimizes for pairwise ranking, commonly used in implicit feedback scenarios.
   * **Logistic**: Suitable for explicit feedback.
   * **WARP-kos**: A variant of WARP for use with highly sparse datasets.
3. **Scalability**: Designed to efficiently handle large datasets.
4. **Ease of Use**: Provides a simple and intuitive API for model training and evaluation.

### **Components of a LightFM Model**

1. **Interactions Matrix**: Represents user-item interactions. In our case, it will be a sparse matrix where rows represent users and columns represent items, and the values represent interactions (e.g., purchases).
2. **User and Item Features**: Optional matrices that include additional information about users and items. For this lab, we will incorporate item features to build a hybrid model.
3. **Loss Function**: Defines how the model is trained. We will experiment with different loss functions to optimize our recommendations.

## 

## **Step-by-Step Guide**

### **Step 1: Data Exploration & Understanding**

Objective: Get familiar with the H&M dataset structure and characteristics.

Key Questions to Explore:

* What does the interaction data look like? How many unique users and items do we have?
* What's the sparsity of the dataset? (Compare total possible interactions vs actual interactions)
* How are interactions distributed across users and items? Are there power users or blockbuster items?
* What time period does the data cover? Are there seasonal patterns?
* What metadata is available for items and customers?

Suggested Analyses:

* Plot distribution of interactions per user and per item (histograms, box plots)
* Identify the long tail: what percentage of items/users account for 80% of interactions?
* Examine the most and least popular items - what patterns do you notice?
* Think about: How might these patterns affect your recommendation strategy?

### **Step 2: Data Sampling Strategy**

Objective: Create a manageable dataset for experimentation while preserving important characteristics.

Why Sample?: Full datasets can be computationally expensive for experimentation. Smart sampling helps you iterate quickly.

Sampling Considerations:

* Should you sample users, items, or interactions? What are the trade-offs?
* How can you maintain the distribution characteristics of the original data?
* Consider sampling strategies: random, stratified, or based on activity levels
* Recommended: Start with active users (e.g., users with 5+ interactions) and popular items

Experiment: Try different sample sizes (1K, 10K, 50K interactions) and observe how it affects model performance.

### **Step 3: Data Preprocessing & Matrix Construction**

**Matrix Construction = best practices le faire comme LightFM l’implement. ON passe des IDs Article/Customer vers des index de 0 à 1 LabelingEncoder dans skitlearn.**

Objective: Transform raw data into formats suitable for LightFM.

Key Tasks:

* ID Mapping: Create integer mappings for user and item IDs (LightFM requires integer indices)
* Interaction Matrix: Build a sparse user-item matrix
  + More info [here](https://making.lyst.com/lightfm/docs/examples/dataset.html), under the sections “Building the ID mappings” and “Building the interactions matrix”.
* Data Cleaning: Handle duplicates, outliers, or invalid entries

### **Step 4: Train/Test Split Strategy**

Objective: Create robust evaluation setup that simulates real-world scenarios.

LightFM doc: [Cross-validation](https://making.lyst.com/lightfm/docs/cross_validation.html)

Splitting Strategies to Consider:

* Temporal Split: Use time-based splits (more realistic for recommendation systems)
* Random Split: Split interactions randomly for each user
* User-based Split: Hold out some users entirely for testing

### **Step 5: Model Training**

Objective: Build and understand a basic collaborative filtering model.

LightFM doc: [model class](https://making.lyst.com/lightfm/docs/lightfm.html)

Experimental Setup:

* Start with a simple model using only interaction data (no features)
* Try different loss functions: WARP, BPR, logistic - what works best for your data?
* Experiment with different numbers of latent factors (dimensions)

Parameters to Explore:

* no\_components: Start with 30-50, experiment with more
* loss: Begin with 'warp' for implicit feedback
* learning\_rate: Try values between 0.01-0.1
* epochs: Monitor convergence (start with 10-20)

### **Step 6: Hyperparameter Optimization**

Objective: Systematically improve model performance.

Approach Options:

* Manual Grid Search: Try combinations of key parameters
* Random Search: More efficient than grid search for many parameters
* Validation-based: Use a separate validation set or cross-validation

Key Parameters to Tune:

* Number of latent factors
* Learning rate and regularization
* Loss function choice
* Number of epochs (watch for overfitting)

### **Step 7: Model Evaluation & Interpretation**

Objective: Assess model quality using appropriate metrics.

LightFM doc: [Evaluation](https://making.lyst.com/lightfm/docs/lightfm.evaluation.html)

Metrics to consider:

* Precision@K & Recall@K: How many relevant items in top-K recommendations?
* AUC: Overall ranking quality
* NDCG: Considers ranking order of relevant items

Evaluation Questions:

* How does performance vary with K (top-5 vs top-20)?
* Are there differences in performance across user segments (active vs casual users)?
* What does "good" performance look like for your use case?

Beyond Numbers:

* Manually inspect recommendations for a few users - do they make intuitive sense?
* Check for diversity: are you recommending only popular items?

### **Step 8: Hybrid Model with Item Features**

Objective: Incorporate item metadata to improve recommendations, especially for cold-start items.

Build item features (based on the [dataset class](https://making.lyst.com/lightfm/docs/examples/dataset.html)) and train a hybrid model

Hybrid Model Experiments:

* Compare pure collaborative filtering vs hybrid model performance
* Test cold-start scenarios: how well does the model recommend new items?
* Feature ablation: which features contribute most to performance?