Contents

[Data 2](#_Toc184507042)

[Files 2](#_Toc184507043)

[Deep Dive 2](#_Toc184507044)

[Methods 7](#_Toc184507045)

[Step 1: Data Loading and Initial Processing 7](#_Toc184507046)

[Step 1.2 Image Embedding 8](#_Toc184507047)

[Step 2: Data Splitting 9](#_Toc184507048)

[Step 3: Feature Engineering Core Metrics 10](#_Toc184507049)

[Step 4: Model Data Preparation 11](#_Toc184507050)

[Step 5: Model Data Preparation 13](#_Toc184507051)

[Step 6: Inference 15](#_Toc184507052)

[Step 7: Evaluation 18](#_Toc184507053)

[Results/Evaluation 19](#_Toc184507054)

Data

## Files

**images/** - a folder of images corresponding to each article\_id; images are placed in subfolders starting with the first three digits of the article\_id; note, not all article\_id values have a corresponding image.

**articles.csv** - detailed metadata for each article\_id available for purchase

**customers.csv** - metadata for each customer\_id in dataset

**transactions\_train.csv** - the training data, consisting of the purchases each customer for each date, as well as additional information. Duplicate rows correspond to multiple purchases of the same item.

## Deep Dive

**Article/Item Data:**

The articles dataset contains detailed information about fashion items, including unique identifiers (article\_id, product\_code), product categorization (product\_type, product\_group, department, section, garment\_group), color information at multiple levels (color\_group, perceived\_color\_value, perceived\_color\_master), visual attributes (graphical\_appearance), and a text description. Each of the 105,542 items has multiple hierarchical classifications, allowing for both broad and granular categorization of fashion products.

A graph with blue squares

Description automatically generated with medium confidence

A graph of a bar graph

Description automatically generated

**Customer Data:**

The customers dataset contains demographic and engagement information for 1.37 million customers. It includes unique customer IDs, membership status (club\_member\_status), fashion newsletter preferences (fashion\_news\_frequency), age, postal code, and two indicators (FN and Active) with significant missing values. This data provides basic customer profiling and their engagement level with the fashion retailer.A graph of age distribution

Description automatically generated

A diagram of a group of blue squares

Description automatically generated

A screenshot of a graph

Description automatically generated

**Transactions Data:**

The transactions dataset records customer purchases, containing the timestamp of purchase (t\_dat), customer and article identifiers that link to their respective datasets, the price of the item, and the sales channel through which the purchase was made (likely indicating online vs in-store purchases). This forms the core interaction data between customers and products.

Transaction Date Range:

First transaction: 2018-09-20 00:00:00

Last transaction: 2020-09-22 00:00:00

A graph of a financial report

Description automatically generated with medium confidence

A graph with blue bars

Description automatically generated

Methods

## Step 1: Data Loading and Initial Processing

The data loading phase forms the foundation of our fashion recommendation system. Working with H&M's fashion retail dataset, we built a pipeline that pulls in customer purchase history, product information, customer profiles, and product images. The biggest challenges we faced were handling large datasets efficiently, processing high-dimensional image data, and ensuring data consistency across different sources. Our solution uses a combination of batch processing, strategic data type optimization, and parallel loading techniques to create a robust foundation for the recommendation engine. This step was crucial because clean, well-structured data directly impacts the quality of our recommendations.

**Core Data Sources**

**Transaction Data**

* Contains the heart of our recommendation data: who bought what and when
* Key fields: customer\_id, article\_id, t\_dat (purchase date), price, sales\_channel\_id
* Historical purchase records help us understand customer preferences and buying patterns

**Customer Data**

* Demographic and customer-specific information
* Includes fields like age, postal\_code, and activity indicators
* Requires careful handling of sensitive information
* Contains both categorical and numerical data

**Article (Product) Data**

* Detailed information about each fashion item
* Product categories, specifications, and attributes
* Price points and product groupings
* Mix of categorical and numerical features

**Image Embeddings**

* ResNet152 embeddings for product images
* 2048-dimensional vectors capturing visual features
* Stored as individual .npy files for efficient loading
* Critical for understanding visual similarity between products

**Implementation Details**

**Primary Loading Mechanism**

We implemented a robust loading system that:

1. Scans a specified directory for CSV files
2. Loads each file into a pandas DataFrame
3. Maintains a dictionary mapping file names to DataFrames
4. Handles errors gracefully without crashing

**Data Type Standardization**

One of our key challenges was ensuring consistent data types across all datasets:

* ID Columns: Converted to strings for consistent matching
* Numerical Columns: Appropriate integer or float types to optimize memory
* Date Columns: Converted to datetime format for temporal analysis
* Categorical Columns: Properly encoded for later processing

**Image Embedding Processing**

The image embedding pipeline involves:

1. Loading individual .npy files containing ResNet152 features
2. Converting high-dimensional embeddings to manageable format
3. Applying PCA reduction while maintaining 95% variance
4. Storing processed embeddings for efficient reuse

## Step 1.2 Image Embedding

We implemented an image embedding pipeline using ResNet152 to convert product images into numerical representations. This process creates dense feature vectors (2048-dimensional) that capture visual characteristics of the fashion items, which are then used in our recommendation system.

**Core Components**

**Model Configuration:**

* Base Model: ResNet152

Configuration:

* Pre-trained on ImageNet
* Top layer removed
* Average pooling
* Output: 2048-dimensional embeddings

**Image Processing Pipeline:**

**Input specifications**:

* Target size: 224 x 224 pixels
* Preprocessing using ResNet's preprocess\_input
* Batch size: 64 images

**Process flow:**

* Load and resize image
* Convert to array
* Expand dimensions
* Apply preprocessing

**Processing System:**

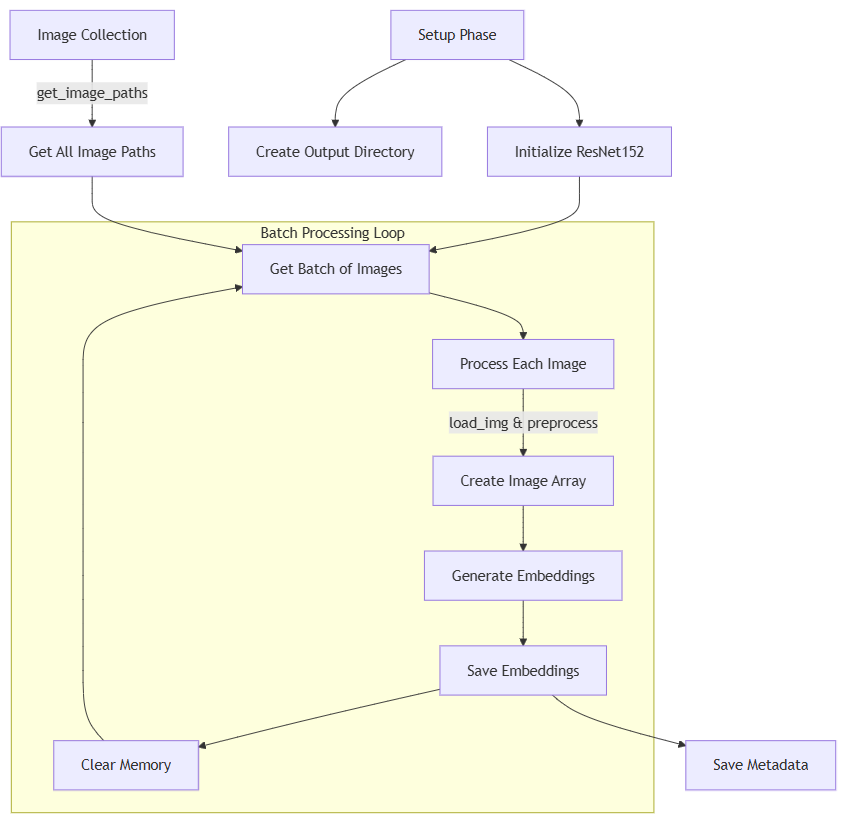
**Process flow:**

* Get model
* Load image paths
* Process images in batches
* Save embeddings
* Memory handling:
* Batch processing
* Memory cleanup after batch
* Individual file saving

**Output Management:**

Storage format:

* Individual .npy files
* Filename matches original image ID
* Each file contains 2048-dimensional vector



## Step 2: Data Splitting

In recommendation systems, properly splitting data is crucial - especially when dealing with fashion retail where trends and purchasing patterns change over time. Rather than using random splitting, we implemented a temporal split that respects the chronological nature of purchases. This means we train on older data and test on newer data, mimicking how the system would work in real life. This approach helps us better understand how our model will perform with future purchases and seasonal trends.

**Temporal Split Implementation**

* Chronologically ordered all transaction data by purchase date
* Created three distinct time periods: training (70%), validation (15%), and test (15%)
* Each split represents a continuous time period
* Later time periods are used for validation and testing
* Ensures no future data leaks into training

## Step 3: Feature Engineering Core Metrics

Feature engineering transforms our raw data into meaningful signals that our recommendation model can learn from. We developed three parallel processing streams - one each for customers, articles, and their interactions. Each stream creates rich features that capture different aspects of the recommendation problem: customer behavior and preferences, article characteristics and popularity, and the dynamics of how customers interact with products. This phase is crucial because the quality of our features directly impacts how well our model can learn purchase patterns.

**Core Metric Calculations**

**1.Customer Features**

We calculate several types of customer metrics, focusing on purchase behavior and recent activity:

**Basic Purchase Behavior**

* Total number of transactions per customer
* First and last purchase dates
* Average purchase price
* Price standard deviation
* Minimum and maximum prices paid
* Total spending
* Count of unique items purchased
* Most frequently used sales channel

**Activity Metrics**

* Days active (calculated as last\_purchase\_date - first\_purchase\_date)
* Purchase frequency (purchase\_count / (days\_active + 1))

**Recent Activity Window (Last 30 Days)**

* Number of recent purchases
* Recent average purchase price
* Recent total spending

**2.Article Features**

Our article features focus on both overall and recent popularity:

**Overall Popularity Metrics**

* Total number of sales
* First and last sale dates
* Number of unique customers
* Average price
* Price standard deviation
* Median price
* Most common sales channel

**Recent Performance (30-Day Window)**

* Number of recent sales
* Number of recent unique customers

**3.Interaction Features**

For each transaction, we capture temporal aspects:

* Day of the week
* Month
* Day
* Weekend indicator (binary flag for Saturday/Sunday)

## Step 4: Model Data Preparation

Before feeding our engineered features into the two-tower recommendation model, we need to transform the data into a format the model can efficiently process. This preparation phase involves three key components: generating negative samples to help the model learn what customers don't like, preprocessing features for consistency, and creating optimized TensorFlow datasets for training. This step is crucial because it directly impacts how well our model learns from the data.

**Core Implementation Components**

**Negative Sampling**

Our negative sampling strategy helps the model learn to distinguish between items a customer might like versus those they wouldn't:

1. Positive Samples

* Uses actual customer-article purchase interactions
* Maintains the original transaction context
* Labels these interactions as positive (1)

1. Negative Sample Generation

* Creates 4 negative samples for each positive interaction
* Randomly samples from available articles
* Labels these interactions as negative (0)
* Preserves customer features from original interaction

1. Combined Dataset

* Merges positive and negative samples
* Maintains proper ratio (1:4)
* Shuffles data to prevent learning bias

**Feature Preprocessing**

1. Customer Features

* Standardizes continuous features (purchase counts, prices)
* Encodes categorical variables
* Converts customer IDs to integer indices
* Handles any missing values

1. Article Features

* Standardizes numerical features
* Processes categorical product attributes
* Converts article IDs to integer indices
* Maintains embedding features from ResNet152

1. Interaction Features

* Processes temporal features
* Maintains original transaction context
* Ensures consistency with both customer and article features

**TensorFlow Dataset Creation**

1. Dataset Structure

* Creates structured format for two-tower model
* Separates customer and article features
* Includes interaction labels
* Maintains batch organization

1. Performance Optimization

* Implements batch size of 256
* Uses shuffle buffer of 10000
* Enables prefetching for efficiency
* Optimizes memory usage

1. Training Configuration

* Structures data for combined loss function
* Separates features for each tower
* Maintains proper feature alignment
* Enables efficient training process

## Step 5: Model Data Preparation

Our recommendation system uses a **two-stage approach: first finding potential matches (retrieval) and then ranking them (ranking).** The architecture **uses two separate neural networks (towers) - one for customers and one for articles - that learn to map each into the same embedding space.** This approach allows us to efficiently find relevant items for each customer by comparing these embeddings.

**Core Components**

* 1. **Retrieval model:**

**Customer (Query) Tower:**

* Input: Customer feature dimension (derived from engineered features)
* Hidden layers: (128, 64)
* Final embedding: 32 dimensions
* All layers use ReLU activation
* Final layer normalization

**Article (Candidate) Tower:**

* Input: Article feature dimension (derived from engineered features)
* Hidden layers: (128, 64)
* Final embedding: 32 dimensions
* All layers use ReLU activation
* **Final layer normalization**

1. **Ranking Model:**

* Customer branch:
  + Dense(64, ReLU)
  + Dense(32, ReLU)
* Article branch:
  + Dense(64, ReLU)
  + Dense(32, ReLU)
* Combined layers:
  + Dense(32, ReLU)
  + Dense(16, ReLU)
  + Dense(1, sigmoid)

1. **Model Configuration:**

* Retrieval temperature: 0.1
* Batch size: 256
* Optimizer: Adam(0.001)
* Loss: Combined retrieval and ranking loss
* Metrics: AUC, Precision, Recall

**Training architecture:**

**A diagram of a company

Description automatically generated**

## Step 6: Inference

The inference phase implements how we generate actual recommendations for customers. We use a two-stage process where we first retrieve potential candidates efficiently, then rank them carefully, while ensuring we don't recommend previously purchased items.

**Core Components**

1. **Single Customer Recommendations**

* Input:
  + Trained model
  + Customer features
  + Article dataframe
  + Interaction history
  + Number of recommendations (K)
* Process flow:
  + Get customer embeddings
  + Calculate retrieval scores
  + Filter previous purchases
  + Rank remaining candidates
  + Return top K articles

1. **Batch Recommendations**

* Input:
  + Trained model
  + Multiple customer features
  + Article dataframe
  + Interaction histories
  + Number of recommendations (K)
* Process flow:
  + Process customer embeddings in batch
  + Generate candidate pools
  + Filter previous purchases for each customer
  + Rank candidates per customer
  + Return top K articles for each customer

1. **Filtering Logic:**

* Maintains purchase history per customer
* Removes previously bought items
* Gets extra candidates if needed after filtering
* Ensures minimum number of recommendations

**Inference Architecture:**

A diagram of a company

Description automatically generated

## Step 7: Evaluation

Our evaluation framework measures the recommendation system's performance through multiple metrics at different cutoff points (K=10,20,30). The RecommenderEvaluator class handles evaluation and reporting, helping us understand both recommendation accuracy and ranking quality.

**Core Components**

1. **Ground Truth Creation:**

* Input: Validation interactions DataFrame
* Process:
  + Creates customer-to-articles mapping
  + Stores actual purchases as sets
  + Reports ground truth statistics
  + Validates data integrity

1. **Metric Calculation:**

* Precision@K:
  + Number of relevant recommended items / K recommendations
  + Measures recommendation accuracy
  + Applied at K = 10, 20, 30
* Recall@K:
  + Number of relevant recommended items / Total relevant items
  + Measures coverage of customer's interests
  + Applied at K = 10, 20, 30
* NDCG@K:
  + Measures ranking quality
  + Considers position of relevant items
  + Applied at K = 10, 20, 30

1. **Results Generation:**

* Report types:
  + Per-user metrics
  + Average metrics across users
  + Detailed metrics at each K
* Format:
  + Summary DataFrame
  + Includes metric values
  + Shows performance at different K values

# Results/Evaluation

**Training:**

Training Loss, Precision and AUC plotted over 500 epochs.

A graph with blue lines

Description automatically generated

1. **Total Loss**:

* Started around 760 and gradually decreased to around 735-740
* Shows overall model improvement but with high variability
* The decreasing trend slowed down after about 200 epochs

1. **Precision**:

* Fluctuates between 0.85 and 0.925
* Improvement in precision in the beginning, following by stagnant improvement trend over time
* Average precision around 0.875-0.9 is relatively good for recommendation tasks

1. **AUC (Area Under Curve)**:

* Ranges between 0.85 and 0.95
* Shows similar volatility to precision
* Maintains consistently high values (>0.85)
* No significant improvement over later epochs

**Inference:**