IST-691: DEEP LEARNING IN PRACTICE - M001

PROJECT PROPOSAL

Fashion Recommendation System: A Hybrid Deep Learning Approach

Fall 2024

[redacted]

Project Overview

Recommendation systems are essential for personalizing customer experience and driving business growth in e-commerce websites like Amazon and Shopify. Fashion e-commerce companies face unique challenges as they must take into account visual factors such as color and pattern, seasonality and unpredictable fashion trends. Our project addresses these challenges by developing a deep learning-based hybrid recommendation system. We developed a two-stage recommendation system for ranking and retrieval using a two-tower neural network architecture. To incorporate visual features, we generated image embeddings and added them as features to our item dataset, alongside other metadata features

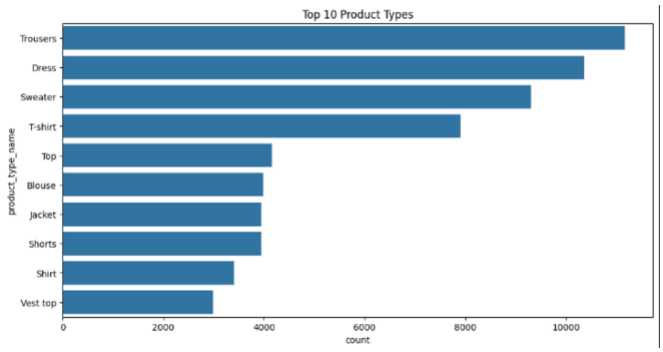
Fashion recommendation systems face unique challenges as they must take into account visual factors such as color and pattern, seasonality and unpredictable fashion trends. Traditional recommendation systems use methods like collaborative filtering, based on user purchase history, or content-based filtering, using product details (Syiam, Wibowo and Setiawan). These methods often miss the nuanced relationships between a product's appearance, user preferences, and fashion trends.

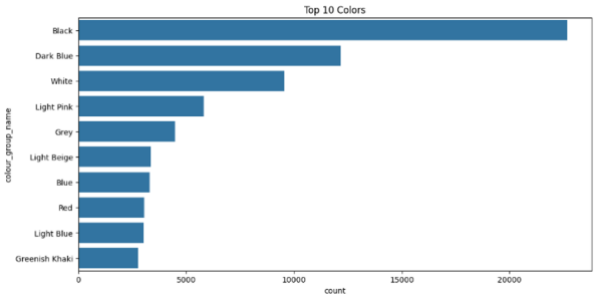
Recent advancements in Deep Learning focus on new hybrid models that combine visual, behavioral, and product metadata features (Zhang, Yao and Sun). Our project leverages these new deep learning techniques to build a two-stage recommendation system that uses a two tower neural network. To incorporate visual features, we used CNNs to generated image embeddings and added them as features to our item dataset, alongside other metadata features

Data

Our data comes from the H&M Kaggle competition and it consists of four main files:
images/ - A folder containing images for each fashion item
articles.csv - metadata for each article available for purchase
customers.csv - metadata for each customer in dataset
transactions_train.csv - training data that contains historical records of customer purchases

Article Data:
The articles dataset contains detailed information about each of the 105,542 fashion items. There are a total of 25 features such as: product categories (product_type, product_group, department, section, garment_group), color information at multiple levels (color_group, perceived_color_value, perceived_color_master), unique identifiers (article_id, product_code), visual attributes (graphical_appearance), and a text description. The columns are a mix of categorical and numeric features. As the product category and color histograms below depict, Trousers and Black colored items have the highest count.

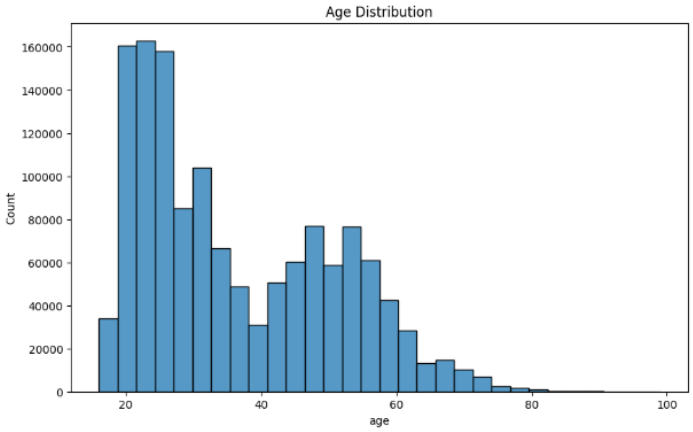




Customer Data:

The customers dataset contains demographic and engagement information for 1.37 million customers. There are a total of 7 features such as: 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.

It is worth noting that personally identifiable customer information like customer_ids and postal codes have been anonymized or transformed into long varchar labels by the creator of the dataset. Regardless, we ensured that customer data was handled carefully because it has sensitive information. The graph below depicts the age distribution of H&M customers. The results show that H&M is popular among people in their 20s which is as expected since that is their target demographic.

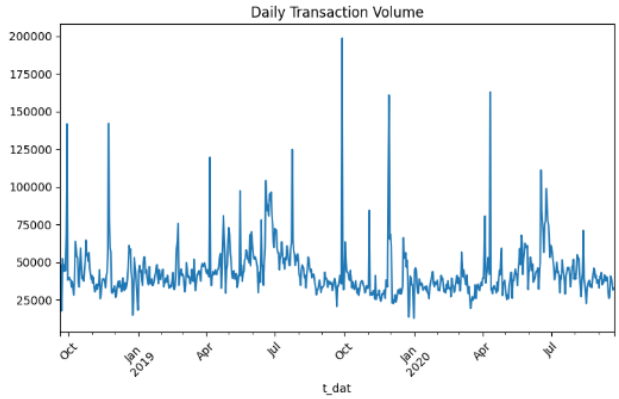


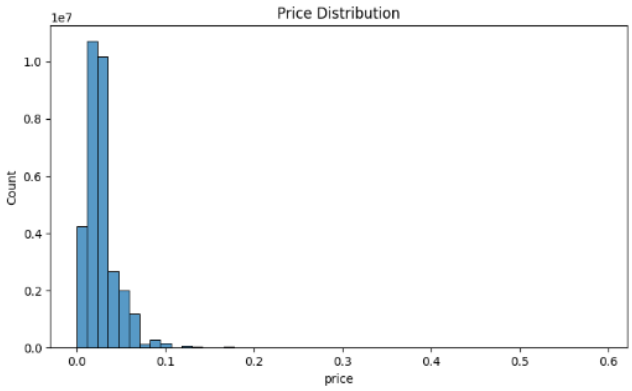
Transactions Data:

The transactions dataset has historical records of 31788324 customer purchases. There are a total of 5 features such as: timestamp of purchase (t_dat), customer and article identifiers that link to their respective datasets, the price of the item, and the channel through which the purchase was made (online vs in-store). Duplicate rows indicate multiple purchases of the same item.

This data is the heart of our recommendation data. It tells us who bought what and when. It helps us understand customer preferences and buying patterns.

The transaction dates range from: 2018-09-20 to 2020-09-22. The line graph below depicts the daily transaction volume and this highlights the dynamic nature of the fashion market. The histogram below shows the price distribution of items sold at H&M. The results show that H&M is a budget friendly fashion retailer, confirming H&M's own branding as a wallet friendly retailer.





Methods:

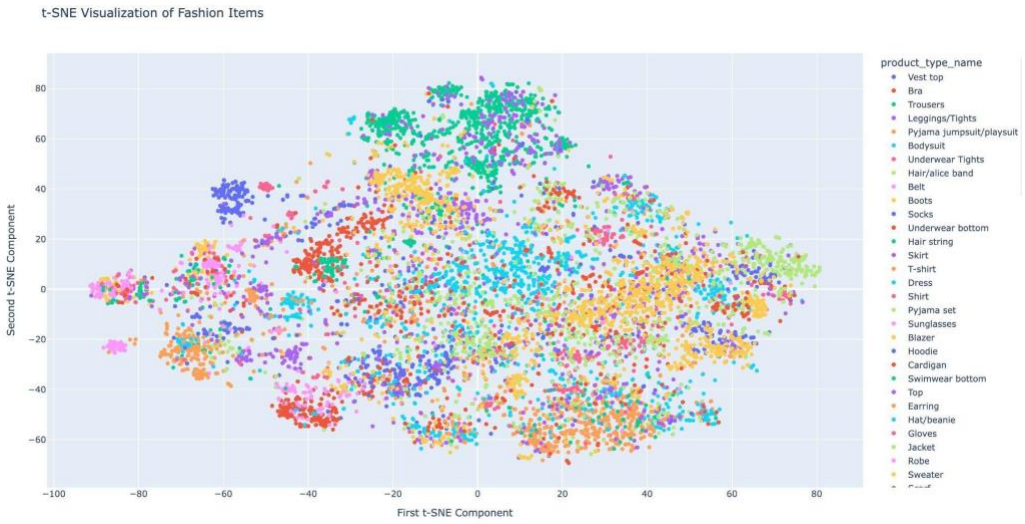
Step 1: Image Embedding

The article metadata features in our dataset do not take into account visual factors such as color, pattern, or the design of a fashion item. To incorporate this visual information into our recommendation process, we embedded the images of each article and added these embeddings as a feature vector to our dataset.

We used ResNet152 which is a deep Convolutional Neural Network (CNN) that is pre-trained on the ImageNet dataset and classifies images into 1000 different categories. The final classification layer of ResNet152 was removed because we're not interested in classifying our images. Then, average pooling was applied to the output of the final convolutional block. This reduces the spatial dimensions, giving us a 2048 dimensional vector that summarizes the visual characteristics of the fashion items. PCA was used to reduce the dimensions of the resulting embeddings from 2048 to 256 while preserving 95% of variance. These embeddings were then added as feature vectors to our dataset, alongside other article attributes.

Why we chose ResNet152: ResNet152 is pre-trained on a large ImageNet dataset. It has already learned useful image representations. This allows us to leverage the useful image representations that it has already learned without needing to train from scratch. This saves computational resources and time. Additionally, ResNet152 overcomes the vanishing gradient problem that deep neural networks face by using skip connections. Skip connections allow information to skip layers in the network. This allows us to train a deep network more effectively.

The resulting image embeddings are visualized in the graph below using t-SNE. As you can observe, all the green dots that represent trousers are clustered together at the top of the graph.



Step 2: Feature Engineering

Our initial dataset has a very limited set of features like transaction dates or product categories. The features did not have any meaningful insights that the model could use to learn the complex relationships between customers and items. To address this, we performed extensive feature engineering to calculate interesting metrics that would capture complex relationships between customers and products.

Customer Features

We calculated several types of metrics for customers and they fall under 2 broad categories: basic purchase stats and customer's activity.

Basic Purchase Stats: To account for a customer's purchase behavior we created the following new features: 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, and Most frequently used sales channel

Customer's Activity: These are the activity metrics that we calculated: Days active (calculated as last_purchase_date - first_purchase_date), Purchase frequency (purchase_count / days_active), Number of recent purchases, Recent average purchase price, and Recent total spending. The last 30 days was considered to be Recent.

Article Features

The features we engineered for articles focus on overall and recent popularity of an article.

Overall Popularity: Total number of sales, First and last sale dates, Number of unique customers, Average price, Price standard deviation, Median price, Most common sales channel, Number of recent sales and Number of recent unique customers. The last 30 days was considered to be Recent.

Interaction Features

For each transaction, we added: Day of the week, Month, Weekend indicator (binary flag for Saturday/Sunday)

Step 4: Data Preparation

Negative Sampling
Negative Sampling is a necessary step when building recommendation systems (Roizner) because our data only tells us what items a customer interacted with (positive samples). It doesn't explicitly tell us what items a customer did not interact with. Negative sampling addresses this issue by generating negative examples. This allows the model to learn patterns in both customer preferences and dislikes.

Our negative sampling strategy generates 4 negative samples for every positive sample. The positive sample is labeled as 1 (positive sample). Then it picks 4 random items that the customer has not interacted with before and labels them as 0 (negative sample). The customer features (e.g. demographics, activity metrics) from the original positive sample are retained and added to these 4 negative samples as well. In the end, the positive and negative samples are shuffled and combined into one dataset.

Temporal Splitting
Temporal splitting is a necessary step in recommendation systems, especially fashion recommendation systems. Random train-test splits will not preserve the chronological nature of purchases. Fashion recommendation systems need to take into account the trends and patterns that change over time.

We want to train on older data and test on newer data, mimicking how the system would work in real life. To implement this temporal split, we ordered all our transaction data chronologically by purchase date. We created three splits or time periods: training (70%), validation (15%), and test (15%). Each split is a continuous time period and later time periods were used for validation and testing.

Feature Preprocessing
We performed some routine preprocessing steps to get our data into a format that the model can process. The steps included: standardizing continuous features (purchase counts, prices), encoding categorical variables, converting to datetime and maintaining embedding features from ResNet152. Null values in numerical columns were filled with the median, while those in categorical columns were replaced with the mode. Additionally, we created TensorFlow datasets so that our data could be in the structure that a TensorFlow Recommender System (TFRS) two-tower model could ingest.

Step 5: Model

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. We used TensorFlow Recommenders (TFRS) ("TensorFlow Recommenders") to create our retrieval and ranking models.

Retrieval model: this model quickly narrows down a large catalog of articles to a smaller set of relevant candidates. The retrieval model consists of a Customer (Query) Tower and an Article (Candidate) Tower that take in customer and article features and encodes them into the same lower dimensional embedding space using a neural network. Relevant articles are retrieved based on similarity between customer and item embeddings. The retrieval model uses Contrastive Loss which teaches a model to tell if two things are similar or not. It does so by comparing similarity scores for customer-item pairs and creating embeddings that are closer together for relevant customer-item pairs.

Customer (Query) Tower: The CustomerTower class from our code defines the customer encoder/ tower. It takes in customer features as input. It consists of fully connected dense layers (128, 64) with ReLU activation and an output layer that projects the customer features into a 32 dimension embedding. Lastly, there is a final normalization layer that normalizes the embeddings.

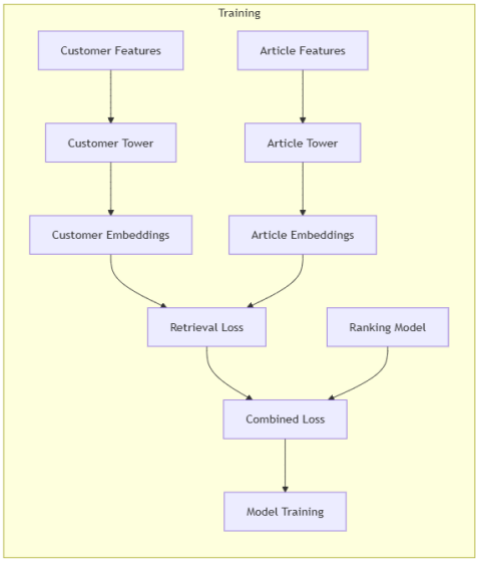
Article (Candidate) Tower: The ItemTower class from our code defines the article encoder/tower. It takes in article features as input. It consists of fully connected dense layers (128, 64) with ReLU activation and an output layer that projects the customer features into a 32 dimension embedding. Lastly, there is a final normalization layer that normalizes the embeddings.

Ranking Model: The RankingModel class in our code defines a model that predicts the probability of interaction between a customer and an article (Biarnes). The RankingModel class in our code has two sub-networks: a customer branch and an article branch (defined as customer_sequential and article_sequential in the code). These two sub-networks process customer and article features independently into smaller representations. Both these branches have two dense layers (64, 32) with ReLU activation.

Then, there is a final network (defined as 'rating' in our code) that concatenates the output of the two branches and predicts a relevance score. This final network consists of two Dense layers (32,16) with ReLU activation and one final Dense layer with one neuron and sigmoid activation function. This final layer outputs a probability between 0 and 1 that a said customer interacts with the article. It uses binary cross entropy loss for interaction prediction. The ground truth of whether a customer has interacted with an item or not comes from the customer's purchase history.

TwoStageRecommender: in our code, this class combines ranking and retrieval into one unified step (Biarnes). It combines ranking loss and retrieval loss to train the model. This leverages multi-task learning to optimize for two objectives: retrieval and ranking. This ensures that the recommendation system retrieves relevant articles and ranks them effectively. Both the retrieval and ranking model train simultaneously (not sequentially) using Adam optimizer with 0.001 learning rate and AUC, Precision and Recall as the metrics.

Training architecture:



Step 6: Inference

The inference stage is where our trained model is used to generate recommendations for users. In the previous steps, our model has been trained on historical data and it learned patterns and relationships from that historical data. Now, the inference stage focuses on using that trained model to provide actual recommendations for customers. We use a two-stage process where we first retrieve potential items and then rank them carefully. The inference step also ensures that we don't recommend previously purchased items.

The workflow for the Inference step is as follows: customer and article features are converted into tensors for input. The customer and article tower generate their respective embeddings. A dot product is performed between customer and article embeddings to compute similarity scores. Top article candidates are retrieved based on similarity scores. Articles that have been previously purchased by that customer are excluded (previous purchases are retrieved from interaction_df). If too many articles are filtered out, additional articles are retrieved to make sure that there are enough recommendations. The ranking model computes a relevance score for each retrieved article. The top 30 ranked articles are selected as the final predictions.

Evaluation Metrics

Our evaluation framework measures the recommendation system's performance through multiple metrics at different cutoff points (K=10,20,30). The RecommenderEvaluator class in our code evaluates the quality of our recommendations using metrics like precision, recall, and NDCG. When all three of these metrics are used together, we get a comprehensive view of our model performance.

Ground Truth Creation: The RecommenderEvaluator creates a dictionary of ground truths that it can use to evaluate the recommendations. It uses the interactions_val dataset to map customers to

items that they've actually purchased in the past. It uses this dictionary as ground truth values to compare the recommendations to.

The 3 evaluation metrics used to evaluate the quality of our recommendations:

Precision @ K - This metric measures the recommendation accuracy by quantifying the number of recommended items that are relevant. In other words it measures how many of the top K items we recommended were actually bought.

$$Precision@K = \frac{Number\ of\ relevant\ items\ in\ top\ K}{K}$$

Where,
K = the number of top items you are considering for evaluation
Number of relevant items in top-K = the items that the user has actually purchased

Recall @ K - This metric quantifies the proportion of relevant items that are found in the top-K recommendations. In other words, of all the items the user bought, how many did we recommend?

$$Recall@K = \frac{Number\ of\ relevant\ items\ in\ top\ K}{Total\ number\ of\ relevant\ items}$$

Where,
Total number of relevant items = the total number of relevant items that the customer purchased (i.e., the ground truth)

NDCG@ K (Normalized Discounted Cumulative Gain @K) - It measures the ranking quality by scoring the recommendations based on the order in which the items were recommended. A high NDCG means that the most relevant items are at the top of the recommendation list.
Relevance score of item is a binary value (0 or 1). Relevant items have a score of 1. Suppose the top 5 recommended items have a relevancy score as follows: [1, 0, 1, 0, 1]. Then Discounted Cumulative Gain is:

$$DCG = \sum_{i=1}^{K} \frac{relevance\ of\ item\ i}{log_2(i+1)}$$

$$DCG\ for\ [1,0,1,0,1] = \frac{1}{log_2(1+1)} + \frac{0}{log_2(2+1)} + \frac{1}{log_2(3+1)} + \frac{0}{log_2(4+1)} + \frac{1}{log_2(5+1)}$$
$$= 1.88685$$

NDCG is the normalized version of the above score. It is calculated as follows:

$$NDCG = \frac{DCG}{IDCG}$$

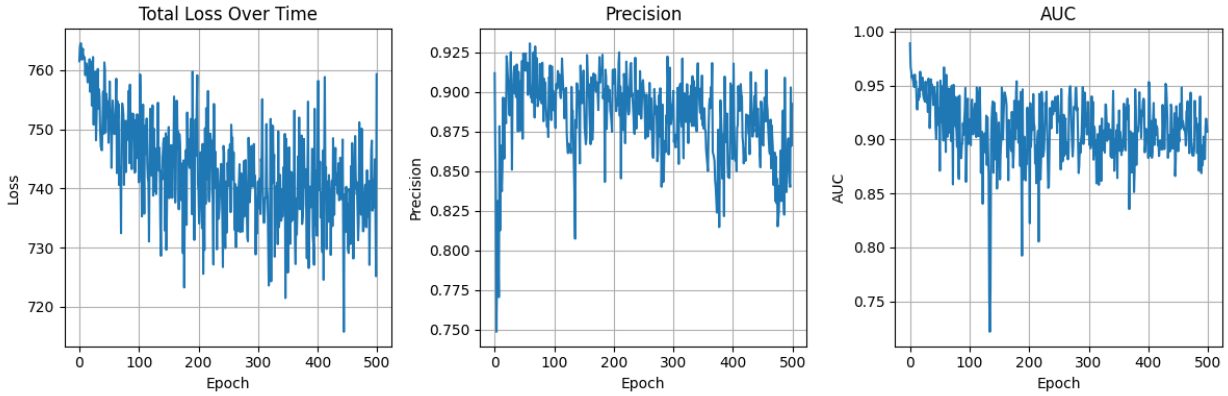
Where IDCG is the idealized DCG where all the top 5 recommended items were relevant, making the relevancy score [1, 1, 1, 1, 1].

Stakeholder Metrics:

From a business perspective, a fashion e-commerce company can track engagement metrics such as Click through Rate and Interactions per user, Revenue metrics such as Conversion rate and Revenue per user and Retention metrics such as User retention rate and Churn rate.

Results

Training Results:



Our model's total loss was around 760 and gradually decreased to around 735-740 after 200 epochs. This indicates that there is an overall improvement in the model as training progresses. However, after the model reached 200 epochs we're seeing diminishing returns with further training. The loss curve has high variability with frequent spikes and dips which means that there is room here to improve and stabilize the learning process through regularization or adjusting learning rate.
Our model's precision improved significantly in the beginning of training and then plateaued. It fluctuates between 0.85 and 0.925 which is relatively good for precision tasks. The AUC curve shows similar behavior to the precision graph, with values fluctuating between 0.85 and 0.95.

Inference Results:

	K=10	**K=20**
Precision@K	60%	55%
Recall@K	0.1%	0.3%
NDCG@K	0.79	0.68

A precision@10 of 60% means that, for a recommendation system, 6 out of the top 10 recommended items are relevant to the user. Precision@K drops slightly to 55% when extending to 20 recommendations. This drop is expected as it becomes harder to maintain precision with more recommendations.

The NDCG scores (0.79 at k=10, dropping to 0.68 at k=20) suggest the system is placing relevant items in good positions, particularly in the top 10 results. In other words, when relevant items are recommended, they are well-placed within the top 10

The low recall (0.1% at k=10 and 0.3% k=20) means that the system is capturing only a small fraction of all relevant items in the top K recommendations. This could be due to several factors:

1. Precision Bias Towards Popular Items: If the system mainly recommends a small subset of popular items precision will be high because those items are often relevant. But Recall will be low because the system isn't recommending from the long list of items that might be unpopular but still relevant. This is a common problem in Fashion recommendation systems because fashion purchases are spread across different categories, making it hard to capture full purchase breadth. Considering this, our recall value isn't too low for a Fashion recommendation system.
2. Cold Start Problem for New Users or Items: New users that lack sufficient interaction history and new items which have not been purchased or interacted with by many customers pose a problem for our model. Recall will be low for these cases.

Works Cited

Syiam, Muhammad, Agung Wibowo and Erwin Setiawan. "Fashion Recommendation System Using Collaborative Filtering." Building of Informatics, Technology and Science (BITS) (2023): 376-385.

Varsha, Moturi. "Two tower recommendation system." 15 May 2024. Medium. 10 November 2024.

Zhang, Shuai, et al. "Deep Learning based Recommender System: A Survey and New Perspectives." ACM Computing Surveys (2018).

Biarnes, Adrien. “Building a Multi-Stage Recommendation System (Part 1.1).” *Medium*, 13 August 2022, https://biarnes-adrien.medium.com/building-a-multi-stage-recommendation-system-part-1-1-95961ccf3dd8. Accessed 12 December 2024.

Biarnes, Adrien. “Building a multi-stage recommendation system (part 1.2).” *Medium*, 25 Aug 2022, https://biarnes-adrien.medium.com/building-a-multi-stage-recommendation-system-part-1-2-ce006f0825d1. Accessed 12 December 2024.

Roizner, Michael. “Two-Tower Networks and Negative Sampling in Recommender Systems.” *Towards Data Science*, 24 November 2023, https://towardsdatascience.com/two-tower-networks-and-negative-sampling-in-recommender-systems-fdc88411601b. Accessed 13 December 2024.

“TensorFlow Recommenders.” *TensorFlow*, https://www.tensorflow.org/recommenders. Accessed 13 December 2024.