Optimization in Preference Learning

Shuhan Zhang Lexuan Chen Xinyu Zhang

May 13, 2025

1 Introduction

1.1 Summary

Our project explores Choice Models and Preference Learning, applying numerical optimization techniques and machine learning algorithms specifically to predict hotel preferences. We aim to accurately quantify consumer preferences, compute choice probabilities, and minimize prediction errors. Using robust methodologies such as the Random Utility Theorem, Multinomial Logit (MNL) models, Frank-Wolfe algorithm, and Matrix Completion techniques, we validate our approaches using Expedia datasets, demonstrating significant practical and theoretical implications.

1.2 Background

Preference learning is a crucial aspect of predictive analytics, widely applied in fields such as personalized marketing and recommendation systems. At its core, it involves modeling consumer choices among various alternatives based on data-driven insights. There are a lot of applications of preference learning, such as Recommendation System, RLHF, Business Analytics, Assortment Optimization and so on. However, real-world preference prediction faces considerable complexity due to large-scale scenarios involving thousands of diverse products and heterogeneous consumer bases. Challenges include partial visibility of options, diverse consumer preferences, and substantial data gaps, making precise modeling an intricate optimization task.

Consider a simplified scenario of predicting consumer choices among fruits based on attributes such as sweetness. Although straightforward on a small scale, scaling this problem significantly amplifies its complexity, necessitating sophisticated optimization techniques to manage effectively.

1.3 Structure

Firstly, we employ the Frank-Wolfe algorithm, specifically addressing the optimization challenges associated with mixture preference models. This approach helps us efficiently manage complex interactions and choice probabilities in large-scale datasets.

Secondly, we apply Matrix Completion techniques, which are crucial for handling extensive missing data scenarios frequently encountered in real-world applications and individual preference learning problems. This method significantly improves our predictive accuracy by effectively estimating unknown preferences from available sparse data.

Through comprehensive experimental validation using Expedia real-world datasets, we illustrate the effectiveness and practical importance of these methodologies in realistic business contexts.

2 Mathematical Formulation of Choice Model

2.1 General Setup

Let N denote the set of all products that a seller can offer to customers. The set of products that the seller actually offers to the customer is denoted as S, where $S \subseteq N$. The choice probability is defined as $P(s|S)$ for $s \in S$, which represents the probability that a customer will choose product s given the offer set S.

2.2 Random Utility Theorem and MNL Model

RUT (Random Utility Theorem) assumes customers map a utility value u^s for each product s, and select the most valuable product in the given offer set, the definition of the mapping can be various.

$$u^s = C^s, u^s = f(x^s), u^s = f(x^s|S)..., \tag{1}$$

where x^s refers to the feature of product s.

To further map the utility value to probability space, Softmax function is adopted, which brings about the multinomial logit model,

$$P(s|S) = \frac{exp(u^s)}{\sum_{s' \in S} exp(u^{s'})} \tag{2}$$

Furthermore, we have mixture MNL model, which assumes that there are different preference types. The choice probability is defined as follow,

$$P(s|S) = \sum_{k=1}^{K} \alpha_k \left(\frac{exp(u_k^s)}{\sum_{s' \in S} exp(u_k^{s'})} \right), \ where \ \sum_{k=1}^{K} \alpha_k = 1 \tag{3}$$

3 Optimization Objective

$$\min_{f \in \mathcal{F}} \quad L(\rho) := -E_S \left[y^\top \log P(s|S, f) \right], \tag{4}$$

where $\mathcal{F}$ is all the possible candidate utility mapping functions from s to u^s, y is the one-hot choice result vector. The objective function is the negative log likelihood loss, which is commonly seen in classification model, but now used in preference learning tasks.

3.1 Model Candidates

1. $u^s = C^s$, **Constant utility**, scenario without feature & small N
2. $u^s = f(x^s)$, **Linear Function** $u^s = w^\top x^s$**, Neural Network**, scenario with feature x^s for product s, w is the weight matrix
3. **Mixture Preference Types**, α_k and u_k^s, $\sum_{k=1}^{K} \alpha_k = 1$, $\alpha_k$is the proportion of preference type k in population
4. ***Matrix Completion, preference inference among individuals based on NN results***

4 Optimization Framework for Mixture Preference Types

Negative Log-Likelihood (NLL) Loss:

$$\text{NLL}(g) = -\frac{1}{N} \sum_{t=1}^{T} \sum_{j \in S_t} N_{jt} \log g_{jt} \tag{5}$$

Likelihood Vector Set: We define the atomic likelihood vector set as

$$\mathcal{F} := \left\{ f(\omega) \in \mathbb{R}^M \mid \omega \in \mathbb{R}^D \right\},$$

where each $f(\omega)$ is the vector of logit choice probabilities computed as

$$f_{jt}(\omega) = \frac{\exp(\omega^\top z_{jt})}{\sum_{\ell \in S_t} \exp(\omega^\top z_{\ell t})},$$

Algorithm 1 Conditional Gradient Algorithm for Estimating the Mixing Distribution

Input: Loss function loss(g), initial guess $g^{(0)} \in \mathcal{F}$, max iteration K
Initialize: $k \leftarrow 0$, set $\alpha^{(0)} = 1$, mixing support $\mathcal{S} = \{g^{(0)}\}$
while stopping condition not met **do**
    $k \leftarrow k + 1$
    Compute gradient $\nabla \text{loss}(g^{(k-1)})$
    Solve support-finding step:

$$f^{(k)} \leftarrow \arg\min_{f \in \mathcal{F}} \langle \nabla \text{loss}(g^{(k-1)}), f - g^{(k-1)} \rangle$$

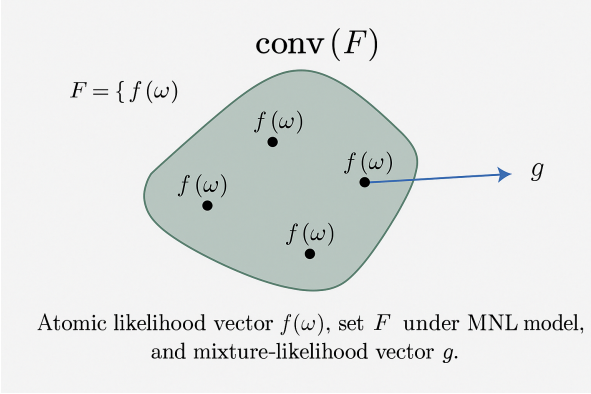
    Update support set: $\mathcal{S} \leftarrow \mathcal{S} \cup \{f^{(k)}\}$
    Fully corrective update:

$$\alpha^{(k)} \leftarrow \arg\min_{\alpha \in \Delta^k} \text{loss}\left(\sum_{s=0}^{k} \alpha_s f^{(s)}\right)$$

    Update solution:

$$g^{(k)} \leftarrow \sum_{s=0}^{k} \alpha_s^{(k)} f^{(s)}$$

end while
Output: Mixing components $\{f^{(s)}\}_{s=0}^{k}$ and weights $\alpha^{(k)}$



Atomic likelihood vector $f(\omega)$, set F under MNL model, and mixture-likelihood vector g.

Figure 1: Likelihood Vector Set

and $f(\omega)$ collects all such $f_{jt}(\omega)$ for (j,t) pairs. This set contains all atomic customer types under the MNL model, and **its convex hull conv($\mathcal{F}$) defines the feasible space of all mixture-likelihood vectors.**

Let the mixing distribution Q be a discrete distribution with support points $\omega_1, \ldots, \omega_K$ and corresponding weights $\alpha_1, \ldots, \alpha_K$, such that:

$$Q = \sum_{k=1}^{K} \alpha_k \, \delta_{\omega_k}, \quad \text{where } \alpha_k \geq 0 \text{ and } \sum_{k=1}^{K} \alpha_k = 1$$

Then the choice probability for product j in offer set S_t is given by:

$$g_{jt} = \sum_{k=1}^{K} \alpha_k \cdot f_{jt}(\omega_k)$$

where

$$f_{jt}(\omega_k) = \frac{\exp(\omega_k^\top z_{jt})}{\sum_{\ell \in S_t} \exp(\omega_k^\top z_{\ell t})}$$

Therefore,

$$g_{jt} = \sum_{k=1}^{K} \alpha_k \cdot \frac{\exp(\omega_k^\top z_{jt})}{\sum_{\ell \in S_t} \exp(\omega_k^\top z_{\ell t})}$$

Support-finding over $\mathcal{F}$ instead of conv($\mathcal{F}$): At each iteration of the conditional gradient (Frank–Wolfe) algorithm, the support-finding subproblem takes the form:

$$\min_{f \in \mathrm{conv}(\mathcal{F})} \langle \nabla \mathrm{loss}(g^{(k-1)}),\ f \rangle,$$

which is a linear minimization problem over the convex hull $\mathrm{conv}(\mathcal{F})$ of atomic likelihood vectors.

By a fundamental result in convex optimization and linear programming, the optimum of a linear function over a convex polytope is always achieved at one of its extreme points (i.e., vertices). Since $\mathrm{conv}(\mathcal{F})$ is defined as the convex hull of $\mathcal{F} := \{f(\omega) \mid \omega \in \mathbb{R}^d\}$, its extreme points lie in $\mathcal{F}$. Therefore, the optimal solution to the support-finding problem lies in $\mathcal{F}$, and we can equivalently solve

$$\min_{f \in \mathcal{F}} \langle \nabla \mathrm{loss}(g^{(k-1)}),\ f \rangle.$$

This fact significantly reduces the computational complexity of each iteration, as it eliminates the need to search over the entire convex hull. In practice, this is leveraged by searching directly over the space of atomic likelihood vectors $f(\omega)$ induced by parameter vectors $\omega \in \mathbb{R}^d$.

Non-convexity of the Support-Finding Subproblem. At each iteration of the conditional gradient algorithm, the support-finding step reduces to the following optimization problem:

$$\min_{\omega \in \mathbb{R}^d} \left\langle \nabla \mathrm{loss}(g^{(k-1)}),\ f(\omega) \right\rangle,$$

where $f(\omega) \in \mathbb{R}^M$ is the atomic likelihood vector computed from the MNL model parameter ω, and each component of $f(\omega)$ takes the form of a softmax probability.

The non-convexity of this problem arises because the mapping $\omega \mapsto f(\omega)$ involves a composition of exponential and rational functions, resulting in a highly nonlinear and non-concave structure. Although the loss function itself is linear in f, the optimization is conducted over the nonlinear transformation of ω, which makes the overall problem non-convex.

Therefore, there is no guarantee of global optimality, and only approximate descent directions $f^{(k)} = f(\omega^{(k)})$ can be found in practice, often by solving the problem via first-order methods such as BFGS.

Neural Network Approximation for Support-Finding. In each iteration of the conditional gradient algorithm, the support-finding step requires solving the optimization problem:

$$\min_{\omega \in \mathbb{R}^d} \left\langle \nabla \mathrm{loss}(g^{(k-1)}),\ f(\omega) \right\rangle,$$

where $f(\omega) \in \mathbb{R}^M$ is the atomic likelihood vector produced by a multinomial logit (MNL) model with parameter ω. As previously discussed, this problem is non-convex and may be difficult to solve directly in high dimensions or under complex feature transformations.

To address this, we propose using a two-layer neural network to approximate the atomic likelihood function $f(\omega)$. Specifically, we parametrize the choice probabilities as:

$$f_\theta(x) = \mathrm{softmax}\left(W_2 \cdot \sigma(W_1 x + b_1) + b_2\right),$$

where $\theta = \{W_1, W_2, b_1, b_2\}$ are the learnable parameters of the network, and $\sigma(\cdot)$ is a nonlinear activation function (e.g., ReLU or tanh).

During each iteration, instead of solving for the optimal logit parameter ω, we optimize the neural network parameters θ to minimize the same linear objective:

$$\min_{\theta} \left\langle \nabla \text{loss}(g^{(k-1)}),\ f_\theta(x) \right\rangle.$$

This formulation allows for greater expressive flexibility, enabling the model to capture more complex patterns in the atomic likelihood structure beyond the linear logit formulation. Additionally, it allows for leveraging modern stochastic gradient optimization techniques for efficient approximation in large-scale settings.

5 Expedia Hotel Dataset

Feature Introduction. The Expedia Hotel Dataset[1] is a publicly available dataset released on Kaggle in 2013. It records customers' search and booking interactions with the Expedia platform. In our experiments, we extract and utilize the subset of features listed in Table 1.

The features can be grouped into two categories: *Hotel Features*, which describe the characteristics of each hotel being recommended, and *Search Criteria*, which describe the preferences or situational context of the customer making the query (e.g., number of adults, length of stay, etc.).

Feature Type	Feature Name	Variable Type	Description
Hotel Feature	PageRank	Discrete	The recommendation position of the hotel
	Star Rating	Continuous	The historical star rating
	Location Score	Continuous	The desirability of a hotel's location
	Historical Price	Continuous	Mean price of the hotel over the last period
	Branded	Binary	Whether the hotel belongs to a major hotel chain
	Promotion	Binary	Whether the hotel has a promotional price
	Price	Continuous	The displayed price of the hotel
	Booking	Binary	Whether the customer booked the hotel
Search Criterion	Booking Window	Discrete	Number of days in the future from the search date
	Length of Stay	Discrete	Number of nights of stay
	Adults Count	Discrete	Number of adults in the party
	Children Count	Discrete	Number of children in the party
	Room Count	Discrete	Number of rooms requested
	Saturday Night	Binary	Whether the stay includes a Saturday night
	Random	Binary	Whether the hotel list is sorted randomly

Table 1: Description of features used in the Expedia Hotel dataset.

Feature Relevance under Different Models. In the context of mixture-of-logit modeling, the effect of different features on utility depends strongly on the expressive capacity of the model.

For **linear logit models**, utility is modeled as an inner product between a weight vector and the hotel's feature vector. Therefore, only features that vary across alternatives (i.e., *Hotel Features*) contribute to utility differentiation and hence affect choice probabilities. In contrast, features that are constant across all alternatives within a given choice set (i.e., *Search Criteria*) have no effect on the relative utility and are effectively ignored during estimation and prediction.

To address this limitation, we incorporate **nonlinear models**—specifically, two-layer neural networks—in the support-finding step. These models are capable of capturing interaction effects between the search context and the hotel features, enabling them to learn how Search Criteria influence preferences in a context-dependent manner. For instance, a longer *Length of Stay* might increase the preference for cheaper hotels or those offering promotions, but such dependencies cannot be modeled with purely linear terms.

Empirically, we observe that the nonlinear model is better able to leverage Search Criterion features, leading to improved predictive accuracy, even when using fewer types in the mixture.

[1] https://www.kaggle.com/datasets/vijeetnigam26/expedia-hotel

6 Experiments

6.1 Experiments Setup

General Setup. To evaluate the effectiveness of our proposed neural network-based support-finding approach, we conduct a comparative experiment against the original linear model baseline described in the Frank–Wolfe framework.

In the baseline setting, we apply the classical Frank–Wolfe algorithm with a linear logit model for the atomic likelihood function and extract a total of 30 customer types. In our proposed method, we replace the linear logit model with a two-layer neural network and limit the number of extracted types to 10.

All architectures share the same optimization settings and are evaluated under the same validation-based early stopping scheme. This design allows us to compare how the expressiveness of the neural network impacts the quality of the recovered mixture model and its generalization performance.

The experiment uses a dataset of 11,000 samples, which is split into:

- **Training set:** 8,000 samples used to run the support-finding iterations;
- **Validation set:** 2,000 samples used to evaluate the model after each new type is added;
- **Test set:** 1,000 samples used to evaluate the final selected model.

During the iterative support construction process, we evaluate the mixture model on the validation set after each new type is added. The model (i.e., the mixture) with the best validation performance is selected as the final model. This selected model is then evaluated on the held-out test set to assess its generalization performance.

Neural Network Architecture. In the neural network-based support-finding approach, we model the atomic likelihood function using a two-layer feedforward neural network followed by a softmax output. The network takes the product feature vector as input and outputs a probability distribution over the offered products, consistent with the multinomial logit framework.

Each network uses the **ReLU activation function** between layers:

$$f_\theta(x) = \text{softmax}(W_2 \cdot \text{ReLU}(W_1 x + b_1) + b_2),$$

where $\theta = \{W_1, W_2, b_1, b_2\}$ are the learnable parameters.

To evaluate the effect of model capacity, we experiment with three hidden layer widths:

- **Large:** 30 hidden units,
- **Medium:** 10 hidden units,
- **Small:** 5 hidden units.

We apply this validation-based early selection procedure to both the baseline and neural network-enhanced versions to ensure a fair comparison under the same data partitioning and evaluation criteria.

Optimization Details. All models share the same optimization setup for consistency and fair comparison. In the **support-finding step**, we solve the non-convex optimization problem over the neural network parameters using the **L-BFGS** optimizer, a quasi-Newton method suitable for smooth loss landscapes. This step is conducted in full-batch mode without data mini-batching, as each support-finding iteration seeks to find a precise direction in the functional space.

For the **proportion-update step**, which optimizes the mixing weights over the convex hull of the discovered atomic likelihood vectors, we employ the **Adam** optimizer with default learning rate and momentum settings. This step is also conducted in full-batch mode for stability and efficiency, given the small dimensionality of the parameter space.

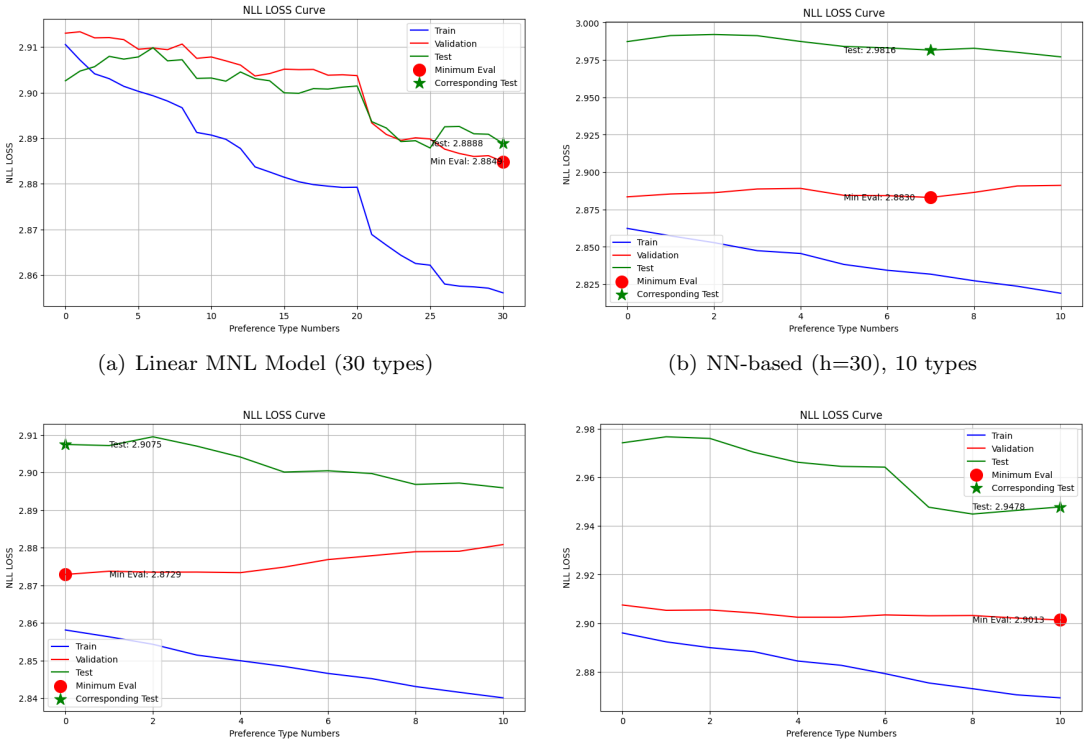
The **initialization phase**, where we obtain the first atomic type via standard MNL estimation or a simple neural estimator, is the only step where **mini-batch training with Adam** is used. This allows the model to scale efficiently with larger datasets during the warm-start phase.

Overall, this training setup is designed to balance efficiency and accuracy, using second-order methods for precise support discovery and first-order methods for scalable and stable weight updates.

Component	Configuration
Dataset Split	8000 training / 2000 validation / 1000 testing
Baseline Model	Linear logit model with 30 types
NN Model Types	2-layer feedforward networks
Hidden Sizes Tested	{30, 10, 5}
Activation Function	ReLU
Atomic Likelihood Function	Softmax over network output
Support Finding Optimizer	L-BFGS (full-batch)
Proportion Update Optimizer	Adam (full-batch)
Initialization	Adam (mini-batch training)
Model Selection	Best validation performance after each type added
Final Evaluation	Best model tested on held-out test set

Table 2: Summary of experimental setup and training configurations.

6.2 Results



(a) Linear MNL Model (30 types)

(b) NN-based (h=30), 10 types

(c) NN-based (h=10), 10 types

(d) NN-based (h=5), 10 types

Figure 2: Comparison of Frank-Wolfe Mixture Models under different architectures.

6.3 Discussion

Although the neural network-based support-finding model is more expressive than its linear counterpart, our experiments reveal that it is highly prone to overfitting—even with a small hidden size and substantial L_2 regularization. This observation holds consistently across all tested configurations.

We hypothesize several possible reasons for this overfitting:

1. **Limited data size.** The Expedia dataset used in our experiments contains only a modest number of training examples. Neural networks typically require large-scale data to generalize

Model	Train	Validation	Test
Linear	2.9119	2.9133	2.8990
NN-30h	2.8616	2.8987	3.2475
NN-10h	2.8647	2.8860	3.1097
NN-5h	2.8641	2.8798	2.9629
Linear - 30 types	2.8561	2.8849	2.8888
NN-30h - 10 types	2.8190	2.8830	2.9816
NN-10h - 10 types	2.8400	2.8729	2.9075
NN-5h - 10 types	2.8692	2.9013	2.9478
Baseline - random guessing	3.5066	3.5066	3.5066

Table 3: Final performance values (NLL, validation, and test) for each model.

well. The lack of sufficient data may lead to poor generalization, regardless of the regularization strength or architecture size.

2. **Under-regularized proportions.** While we applied regularization to the support-finding step (i.e., training each atomic neural network), we did not penalize complexity during the proportion-update step. As a result, complex types may still receive large weights in the mixture even if they overfit the training data. Future work may benefit from incorporating sparsity-inducing penalties or entropy regularization in the weight optimization step.

3. **Inherent linearity of human decision boundaries.** Perhaps the most fundamental reason is that, in real-world decision-making scenarios such as hotel booking, human preferences may inherently follow relatively linear structures. For example, customers typically prefer cheaper hotels, higher star ratings, or more prominent locations. In such settings, linear models already capture most of the signal, and introducing nonlinearity may only add noise and overfitting risk without meaningful gains.

These results highlight a key lesson: *machine learning is not a hammer looking for nails.* The success of a model depends on how well its inductive bias aligns with the underlying structure of the problem. There is no free lunch in modeling—expressiveness alone does not guarantee better performance. In preference learning, modeling assumptions should be grounded in an understanding of how humans make decisions in the specific application context, rather than assuming that more powerful models like neural networks will always perform better.

7 New Trial: Matrix Completion

7.1 Background

In our previous sections, we modeled user choice behavior via utility-based selection rules, producing predictions that aligned well with observed booking behavior. However, when these predictions are organized into a matrix over (user, hotel) pairs, the result is naturally sparse. This sparsity closely resembles the structure of a **matrix completion** problem, where the objective is to recover missing entries from partial observations.

7.2 Model Assumption of Matrix Completion

Matrix completion aims to recover a complete matrix from a small subset of observed entries under the assumption that the underlying matrix is approximately low-rank. This low-rank structure can be enforced either implicitly through convex relaxations (e.g., nuclear norm minimization), or explicitly via low-dimensional matrix factorization. In the latter case, the target matrix $X \in \mathbb{R}^{m\times n}$ is modeled as the product of two lower-dimensional matrices $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{n\times k}$, such that:

$$X \approx UV^\top$$

Here, each row of U represents the latent features of a user, while each row of V represents those of an item. This factorized representation enables compact modeling of user–item interactions, and has been widely adopted in large-scale recommendation tasks — most notably in the context of the Netflix Prize Challenge.

7.3 Construction of the Sparse Utility Matrix

In our setup, the Expedia hotel choice data is used to construct a large and sparse utility matrix. For each user (i.e., each search session), we first generate a personalized *offer set* based on their browsing history. A utility score is then estimated for each hotel in this offer set using a 3-layer feedforward neural network. The network takes as input a joint feature vector of the form x_{ij}, representing the consumer–hotel pair, and outputs a scalar utility score:

$$f(x_{ij}) = W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 x_{ij}))$$

Here, $\sigma(\cdot)$ denotes the sigmoid activation function, and W_1, W_2, W_3 are weight matrices of the neural network. The model architecture is as follows:

- **Layer 1**: $\mathbb{R}^d \to \mathbb{R}^{100}$, followed by a sigmoid activation.
- **Layer 2**: $\mathbb{R}^{100} \to \mathbb{R}^{100}$, with another sigmoid activation.
- **Output Layer**: $\mathbb{R}^{100} \to \mathbb{R}^1$, producing a scalar utility score.

To construct the global utility matrix $X \in \mathbb{R}^{m \times n}$, we define each row to represent a user (search session), and each column to represent a hotel from the union of all users' offer sets. Initially, all entries in X are set as missing. We then populate only the entries X_{ij} for which hotel j appears in user i's offer set, using the predicted utility score $f(x_{ij})$. Since each user typically views only a small fraction of the total hotel pool, the resulting matrix is highly sparse. This sparsity motivates the application of matrix completion techniques to recover the missing entries and improve recommendation quality.

7.4 Optimization Objective.

In our experiments—both on synthetic data and the real-world dataset—our objective is twofold: to accurately recover the missing utility scores while ensuring low prediction error on the observed entries, and to maintain a low-rank structure in the recovered matrix so as to preserve model compactness and mitigate overfitting. To this end, we minimize the following regularized reconstruction loss:

$$\mathcal{L}(U, V) = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \left(X_{ij} - \hat{X}_{ij} \right)^2 + \lambda \left(\|U\|_F^2 + \|V\|_F^2 \right) \tag{6}$$

Here, X_{ij} denotes the observed utility score, $\hat{X}_{ij}$ is the predicted score (including bias terms), λ is a regularization coefficient, and Ω is the index set of observed entries. The prediction matrix is given by $\hat{X} = UV^\top$, and the loss is computed only over observed entries.

To quantitatively assess the reconstruction quality, we evaluate the following standard metrics:

- **Mean Squared Error (MSE)**: average squared prediction error on observed entries.
- **Root Mean Squared Error (RMSE)**: square root of the MSE, providing error in the original scale.
- **Mean Absolute Error (MAE)**: average absolute deviation between predicted and observed values.

7.5 Experimental Path

Our matrix completion experiments were carried out in two stages, with the first focusing on algorithm selection and validation, and the second applying the best-performing method to the Expedia dataset.

Stage 1: Method Evaluation on Synthetic and Benchmark Data. To identify a reliable and scalable matrix completion method, we first tested several techniques on synthetic low-rank matrices. We constructed these by generating two random matrices $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$, then computing $M = UV^\top$. We created the observed matrix X by randomly sampling a fraction ρ of entries and masking the rest:

$$X_{ij} = \begin{cases} M_{ij}, & \text{if } (i,j) \in \Omega \\ \text{NaN}, & \text{otherwise} \end{cases} \tag{7}$$

We evaluated two main categories of methods:

- **SVT (Singular Value Thresholding)** solves a convex relaxation via nuclear norm minimization:

$$\min_X \lambda\|X\|_* + \frac{1}{2}\|X\|_F^2 \quad \text{s.t. } \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \tag{8}$$

  where $\|X\|_*$ denotes the nuclear norm, $\|X\|_F$ is the Frobenius norm, and $\mathcal{P}_\Omega(\cdot)$ is the projection operator that retains only the observed entries. The optimization is performed via an iterative proximal method with singular value soft-thresholding applied at each step. However, traditional SVT methods suffer from two major drawbacks. First, they are **computationally inefficient**: each iteration requires a full singular value decomposition (SVD), which becomes prohibitively expensive for large-scale matrices. Second, the nuclear norm relaxation tends to **overestimate the effective rank** of the solution, often yielding overly dense or high-rank reconstructions that compromise interpretability and reduce generalization. As a result, SVT is less favored in modern recommender system pipelines, where computational efficiency and model sparsity are essential. In light of these limitations, we adopt low-rank matrix factorization techniques as a more scalable and interpretable alternative for matrix completion tasks.

- **Low-rank matrix factorization** with various optimization strategies. To incorporate systematic biases, we include a global bias b, user-specific bias b_{u_i}, and item-specific bias b_{v_j} in the prediction model. The global bias is defined as the mean of all observed entries:

$$b := \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} X_{ij}.$$

  The user and item biases are computed as the deviations from the global mean:

$$b_{u_i} := \frac{1}{|\Omega_i|} \sum_{j\in\Omega_i} X_{ij} - b, \quad b_{v_j} := \frac{1}{|\Omega_j|} \sum_{i\in\Omega_j} X_{ij} - b,$$

  where $\Omega_i = \{j \mid (i,j) \in \Omega\}$ and $\Omega_j = \{i \mid (i,j) \in \Omega\}$ denote the sets of items rated by user i and users who rated item j, respectively. If a user or item has no observed entries, the corresponding bias is set to zero. The **full prediction model** is then defined as:

$$\hat{X}_{ij} = b + b_{u_i} + b_{v_j} + (UV^\top)_{ij} \tag{9}$$

  To simulate realistic noise conditions, we added small Gaussian noise to each gradient update. The following update strategies were compared:

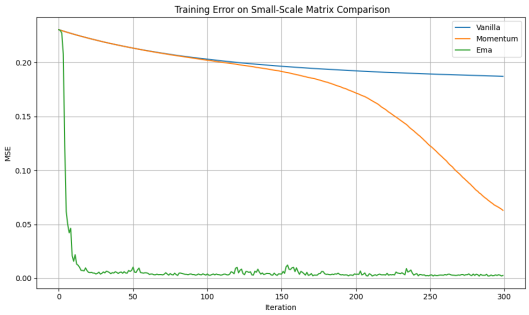
  - **GD**: Standard gradient descent.
  - **Momentum**: Adds a velocity term to accelerate convergence:

$$v_t = \beta v_{t-1} + \nabla L, \quad \theta \leftarrow \theta - \alpha v_t + \varepsilon \tag{10}$$

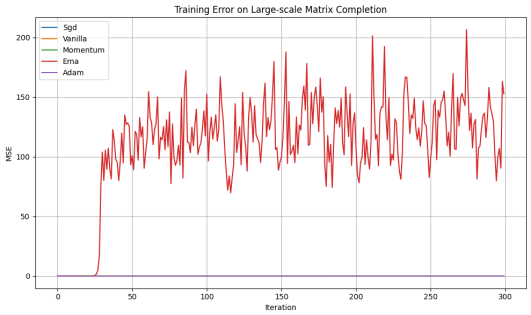
  - **EMA**: Applies exponential moving averages directly to the parameter updates:

$$\theta_{\text{ema}} \leftarrow \theta_{\text{ema}} - \alpha\nabla L, \quad \theta \leftarrow (1-\gamma)\theta_{\text{ema}} + \gamma\theta + \varepsilon \tag{11}$$

  - **Adam**: A optimizer that combines momentum and adaptive learning rate .

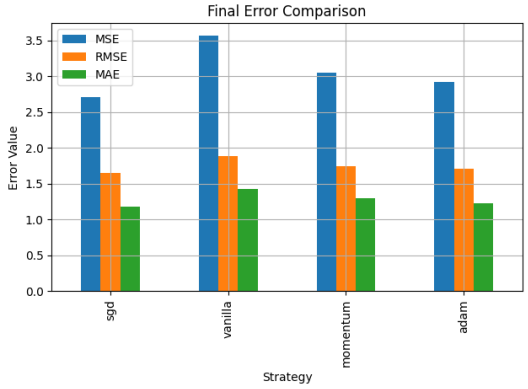


(a) EMA performs best in small-scale setting.



(b) EMA instability on large-scale matrix completion.

(c) Training error across optimizers (noisy setting).



(d) Final MSE, RMSE, and MAE of all optimizers.

Figure 3: Comparison results of different methods in both large and small scales.

To systematically identify a robust optimizer for large-scale matrix completion, we designed a two-phase evaluation. In the first phase, we included all candidate optimal methods—Gradient Descent, Momentum, EMA, and Adam—and conducted tests on synthetic low-rank matrices with injected noisebased with GD without noise.

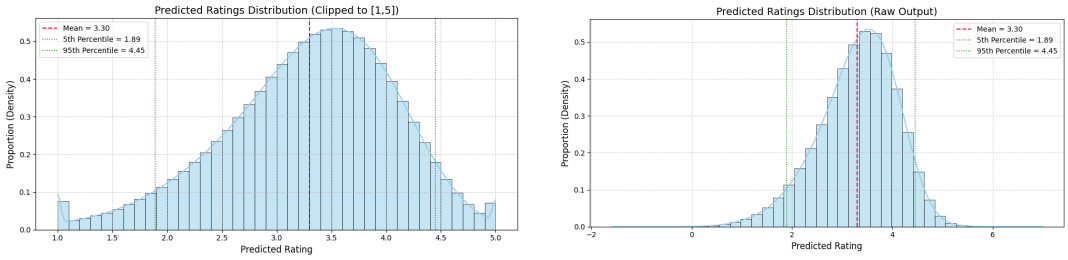
Our initial experiments revealed that the EMA-based optimizer, despite its strong performance in small-scale synthetic settings, led to highly unstable or plateaued behavior when applied to large matrices. This instability was primarily attributed to aggressive gradient accumulation, as illustrated in Figure 3(b). Based on this observation, we excluded EMA from further consideration. In the second phase, we re-evaluated the remaining optimizers—SGD, Momentum, and Adam—under consistent large-scale conditions. Figure 3(c) shows their convergence behavior over 300 iterations, while Figure 3(d) compares their final MSE, RMSE, and MAE metrics. Adam consistently achieved the best balance among convergence speed, reconstruction accuracy, and numerical stability. Momentum offered moderate improvements over vanilla gradient descent, while SGD showed slower but stable descent. These results justify our choice of Adam as the default optimizer in all subsequent experiments, including those on real-world datasets.

Netflix Benchmark We benchmarked our Adam-based matrix factorization on the Netflix dataset against a popular SVT-based notebook from Kaggle. We performed a grid search over k, ℓ_2, learning rate, and loss type (MSE vs. Huber) to tune our hyperparameters, Huber will be formally defined in next section. The top configurations are summarized below:

The original notebook reported a best RMSE of **2.1635** on a test set of 2,500 entries. Our model achieved a significantly better result with RMSE **0.8644** using $k = 30$, learning rate 10^{-3}, MSE loss, and $\ell_2 = 10^{-3}$. Using this best configuration, we next omputed predictions for all user-item pairs. The final predicted distribution is shown in Figure 4, and top-10 movie recommendations are listed in Table 5. This strong performance on a real-world dataset gave us confidence to apply the method to Expedia data.

k	ℓ_2	Loss	LR	RMSE	Epoch
30	0.0010	MSE	0.001	0.8615	197
50	0.0001	Huber	0.001	0.8619	154
30	0.0005	MSE	0.001	0.8619	189
20	0.0001	MSE	0.001	0.8621	220
20	0.0005	Huber	0.001	0.8626	218

Table 4: Top-5 grid search results on Netflix.



(a) Predicted score distribution with clip

(b) Predicted score distribution without clip

Figure 4: Predicted score distribution on Netflix.

Stage 2: Real-World Expedia Data. Motivated by the strong performance on the Netflix dataset, we applied the Adam-optimized matrix factorization on sparse utility data from Expedia.

This stage involves two distinct train-test splits:

1. **Neural Network Model Training.** To produce the utility scores that would populate our matrix, we first trained a Deep-MNL model. This model was trained using 7,000 user sessions from the first half of 2013 and evaluated on a test set of 1,000 sessions from the second half of 2013. These sessions were processed under consistent filtering criteria: only sessions targeting a single destination were retained, and the position feature was removed to ensure the scores were rank-free. This yielded session-level utility predictions (before softmax), which were used to construct the sparse matrix.

2. **Matrix Completion Evaluation.** Using the trained Deep-MNL model, we selected 1,000 hotels at random from the Expedia dataset. We then identified all sessions containing at least one of these hotels, yielding a user-hotel matrix of size 2961×31471, where each entry represents a predicted utility score. In total, 80,931 entries were non-missing (observed), forming the base matrix for matrix completion. From this, we randomly selected 3,000 valid (non-NaN) positions as a test set for evaluation. These values were masked (set to NaN) in the matrix during training and retained as ground truth for test RMSE computation.

Rank	Title	Predicted Rating
1	Song of Freedom (1936)	5.17
2	Lured (1947)	5.07
3	Brother of Sleep (1995)	5.06
4	Gate of Heavenly Peace, The (1995)	5.03
5	Follow the Bitch (1998)	5.03
6	Mamma Roma (1962)	5.02
7	One Little Indian (1973)	4.99
8	Bittersweet Motel (2000)	4.90
9	Smashing Time (1967)	4.84
10	Baby, The (1973)	4.82

Table 5: Top-10 movie predictions from matrix completion on Netflix.

Additional Optimization Objective Given the extreme sparsity observed in the Expedia matrix, we experimented with multiple loss functions to improve robustness. In addition to Mean Squared Error (MSE), we evaluated the use of Huber loss, which is known for its resilience to outliers and noisy targets. Huber loss behaves quadratically for small errors and linearly for large errors, offering a smooth trade-off between MSE and MAE:

$$\mathcal{L}(U,V) = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \text{Huber}(\hat{X}_{ij} - X_{ij}, \delta) + \lambda \left(\|U\|_F^2 + \|V\|_F^2 \right) \tag{12}$$

$$\text{Huber}(x,\delta) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \le \delta \\ \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{13}$$

7.6 Results.

Empirically, grid search results showed that Huber consistently outperformed MSE in terms of final RMSE:

k	ℓ_2	Loss	LR	RMSE	Epoch
30	0.0001	Huber	0.01	0.6109	27
30	0.0010	Huber	0.01	0.6132	26
20	0.0010	Huber	0.01	0.6145	30
30	0.0005	Huber	0.01	0.6149	27
20	0.0001	Huber	0.01	0.6158	31

Table 6: Top-5 configurations using Huber loss on Expedia data.

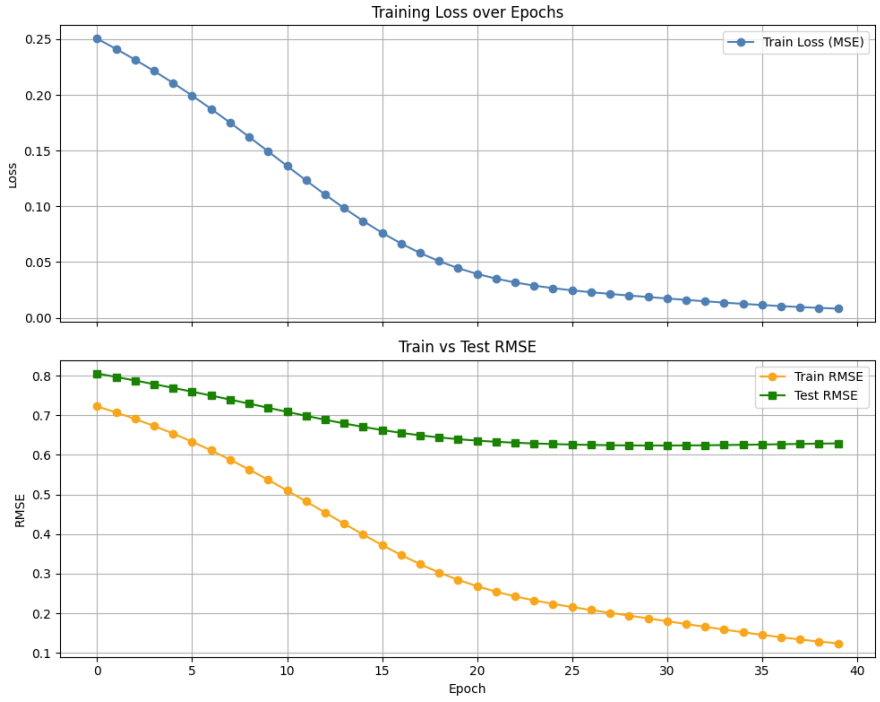


Figure 5: Training loss and RMSE curves under best-performing configuration (Huber loss, $k = 30$).

Discussion on Early Stopping. Our training was equipped with early stopping based on validation RMSE plateau. Interestingly, in the Expedia experiment, early stopping was triggered as soon as epoch

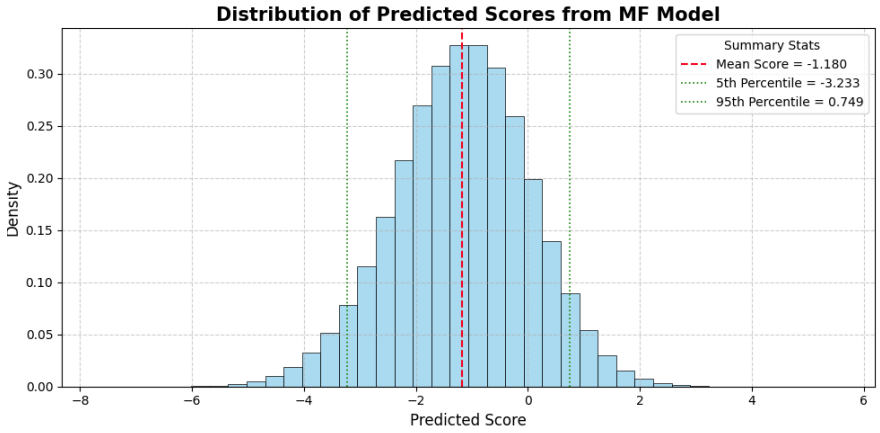


Figure 6: Distribution of predicted utility scores from matrix completion (Expedia).

27. This suggests that the model was able to capture most of the meaningful signal early in training and that further optimization would risk overfitting due to the matrix's extreme sparsity and noisy target structure. Moreover, the presence of learned user and item biases might have absorbed substantial variation, allowing latent factors to converge quickly.

Reflections. While our method performed well on both datasets, we recognize limitations in applying matrix completion to Expedia data:

- The constructed matrix is extremely sparse (density ¡ 0.1%), far lower than theoretical guarantees for successful recovery.
- Expedia lacks unbiased ground-truth scores. Utilities are derived from a preference model, not explicit feedback.
- A more principled approach would involve clustering users and items to improve matrix coherence before completion.

Ultimately, our findings underscore the robustness and interpretability of matrix completion techniques as a foundation for preference modeling—even in regimes characterized by extreme sparsity and noisy, proxy feedback. By decomposing observed ratings into latent user-item interactions along with bias structures, matrix factorization offers a granular, individual-level understanding that traditional population-wide preference models often overlook. This individualized lens not only enhances model personalization but also bridges the gap between statistical abstraction and real-world applicability, particularly in scenarios such as cold-start problems, targeted recommendations, and adaptive content delivery. As such, matrix completion serves not merely as an imputation tool, but as a principled framework for deepening the expressiveness and adaptability of modern recommendation systems.

References

- Y. Chi, "Low-rank matrix completion," *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 178–181, 2018.
- J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *arXiv preprint* arXiv:0810.3286, 2008.
- O. Golden, "Netflix Problem – Singular Value Thresholding," *Kaggle Notebook*, https://www.kaggle.com/code/odedgolden/netflix-problem-singular-value-thresholding/notebook