Hyperparameter Optimization for Recommendation Models Using Genetic Algorithms

Matheus Muniz Damasco
math.muniz.damasco@gmail.com
Federal University of Juiz de Fora
Computational Modeling
Juiz de Fora, Minas Gerais, Brasil

ABSTRACT

This paper presents an approach to hyperparameter optimization for recommendation models using genetic algorithms. The focus of this study is on optimizing three key hyperparameters—batch size, learning rate, and latent factors—in two popular recommendation models: Matrix Factorization and a simple Neural Network. Both models were developed using the Keras framework and tested on two well-known datasets, MovieLens 100K and MovieLens 1M from GroupLens. Hyperparameter optimization is a critical aspect of improving model performance, as poorly tuned parameters can lead to sub optimal results. The goal is to minimize the mean absolute error (MAE) after 10 epochs by employing genetic algorithms to effectively explore the hyperparameter space. Our experiments demonstrate significant improvements in MAE, particularly for the Matrix Factorization model. Although our focus was on MAE to understand how the genetic algorithm affected these metrics. These results highlight the effectiveness of genetic algorithms in hyperparameter tuning for recommendation systems and their potential to enhance model performance in real-world applications.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Genetic algorithms**.

KEYWORDS

Hyperparameter optimization, Genetic algorithms, Recommendation systems, Collaborative filtering, Matrix factorization, Neural Networks

1 INTRODUCTION

In today's digital landscape, recommendation systems play a critical role in enhancing user experience and driving business success. Platforms such as Netflix, iFood, Amazon, and Medium leverage recommendation algorithms to boost sales, increase user engagement, and maximize the time users spend on their platforms. These systems rely heavily on techniques like Collaborative Filtering, which provides personalized recommendations by analyzing the preferences of users with similar tastes.

This project focuses on the optimization of three hyperparameters in recommendation models: batch size, learning rate, and latent factors. Hyperparameter tuning is crucial for improving model performance, as it directly influences the learning process and model accuracy. Without proper optimization, even robust models can deliver suboptimal results. In this work, two widely used models Matrix Factorization and a simple Neural Network were implemented using the Keras framework. Both models were trained on the popular MovieLens datasets, which are frequently used in academic research due to their size and diversity, making them ideal for evaluating recommendation models in collaborative filtering tasks.

The primary goal of this study is to apply a genetic algorithm to optimize the aforementioned hyperparameters, with a focus on minimizing the mean absolute error (MAE) after 10 epochs. This analysis helps to understand how tuning hyperparameters not only reduces prediction error but also affects the balance between relevance and diversity in recommendations.

2 PROBLEM DESCRIPTION

The primary problem addressed in this work is the **optimization of hyperparameters** in two recommendation models: **Matrix Factorization** and a **Neural Network**. Hyperparameter tuning plays a vital role in the performance of machine learning models, especially in recommendation systems where personalized suggestions are made based on user preferences. Poorly tuned hyperparameters can lead to suboptimal recommendations, increasing the error and reducing the overall user experience.

This problem is approached as a **single-objective optimization task**, with the goal of minimizing the **Mean Absolute Error (MAE)**, which is a widely used metric in recommendation systems to measure prediction accuracy.

The three key hyperparameters optimized in this study are:

- **Batch Size**: Controls the number of samples processed before the model updates its weights.
- **Learning Rate**: Dictates how much to adjust the weights of the model with respect to the loss gradient.
- **Latent Factors**: In Matrix Factorization, this represents the number of features used to represent both users and items in a lower-dimensional space.

2.1 Constraints and Search Space

The optimization of these hyperparameters was subject to the following constraints:

- **Batch Size**: Values range between 64 and 512.
- **Learning Rate**: Values range between 1×10^{-10} and 1×10^{-2}.
- **Latent Factors**: Values range between 32 and 256.

The hyperparameter search was conducted using **genetic algorithms**, which are well-suited to exploring complex, multi-dimensional

spaces efficiently. The search aimed to minimize the **MAE** after 10 training epochs for both models.

This optimization was applied to two widely-used datasets from Group Lens, **Movie Lens 100K** and **Movie Lens 1M**, which consist of user-item interactions and are standard benchmarks for evaluating recommendation systems.

3 MOVIELENS DATASET

There are several datasets available for recommendation research. Among them, the MovieLens dataset is arguably one of the most popular. MovieLens is a non-commercial, web-based movie recommendation system created in 1997 and maintained by GroupLens, a research lab at the University of Minnesota, to collect movie rating data for research purposes. MovieLens data has been critical for numerous research studies, including personalized recommendations and social psychology.

The MovieLens dataset is hosted by GroupLens and is available in several versions. In this work, we use two versions: MovieLens 100K and MovieLens 1M datasets.

3.1 MovieLens 100K

The MovieLens 100K dataset was released in April 1998 and consists of:

- 100,000 ratings ranging from 1 to 5 stars,
- 943 users,
- 1,682 movies.

3.2 MovieLens 1M

The MovieLens 1M dataset was released in February 2003 and contains:

- 1,000,209 ratings ranging from 1 to 5 stars,
- 6,040 users,
- 3,883 movies.

4 ALGORITHMS

4.1 Matrix Factorization

[1] Matrix factorization is a well-established algorithm in the literature of recommendation systems. The first version of the matrix factorization model was proposed by Simon Funk in a famous blog post, where he described the idea of factorizing the interaction matrix. It later gained widespread recognition due to the Netflix competition held in 2006. At that time, Netflix, a media streaming and video rental company, announced a contest to improve the performance of its recommendation system. The team that could improve the baseline Netflix system, Cinematch, by 10 percent would win a one million-dollar prize. As a result, the competition attracted significant attention in the field of recommendation system research. Eventually, the grand prize was won by the Pragmatic Chaos team from BellKor, a combined team from BellKor, Pragmatic Theory, and BigChaos (you don't need to worry about these algorithms in this project). Although the final score was the result of an ensemble solution, the matrix factorization algorithm played a critical role in the final mix. The technical report of the Netflix Grand Prize solution [2] provides a detailed introduction to the model adopted. In this section, we will dive into the details of the matrix factorization model and its implementation.

4.2 Matrix Factorization Model

Matrix factorization is a class of collaborative filtering models. Specifically, the model factorizes the user-item interaction matrix (e.g., rating matrix) into the product of two lower-dimensional matrices, capturing the underlying structure of the user-item interactions.

Let $\mathbf{R} \in \mathbb{R}^{m \times n}$ be the interaction matrix with m users and n items, where the values of $\mathbf{R}$ represent explicit ratings. The user-item interaction is factorized into a latent user matrix $\mathbf{P} \in \mathbb{R}^{m \times k}$ and a latent item matrix $\mathbf{Q} \in \mathbb{R}^{n \times k}$, where $k \ll m, n$ is the latent factor size. Let $\mathbf{p}_u$ denote the u-th row of $\mathbf{P}$, and $\mathbf{q}_i$ denote the i-th row of $\mathbf{Q}$. For a given item i, the elements of $\mathbf{q}_i$ measure the extent to which the item possesses certain characteristics, such as movie genres or languages. For a given user u, the elements of $\mathbf{p}_u$ measure the extent of the user's interest in the corresponding item characteristics. These latent factors can represent obvious dimensions or may be completely uninterpretable.

The predicted ratings can be estimated as:

$$\hat{\mathbf{R}} = \mathbf{P}\mathbf{Q}^\top$$

where $\hat{\mathbf{R}} \in \mathbb{R}^{m \times n}$ is the predicted rating matrix with the same shape as $\mathbf{R}$. A significant limitation of this prediction rule is that it does not model user/item biases. For example, some users tend to give higher ratings, or some items consistently receive lower ratings due to lower quality. These biases are common in real-world applications. To capture these tendencies, user and item-specific bias terms are introduced. Specifically, the predicted rating that user u gives to item i is calculated by:

$$\hat{\mathbf{R}}_{ui} = \mathbf{p}_u \mathbf{q}_i^\top + b_u + b_i$$

Next, we train the matrix factorization model by minimizing the mean absolute error (MAE) between the predicted ratings and the actual ratings. The objective function is defined as:

$$\underset{\mathbf{P},\mathbf{Q},b}{\operatorname{argmin}} \sum_{(u,i)\in\mathcal{K}} \left|\mathbf{R}_{ui} - \hat{\mathbf{R}}_{ui}\right| + \lambda \left(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2 + b_u^2 + b_i^2\right)$$

where λ is the regularization rate. The regularization term is the $\lambda \left(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2 + b_u^2 + b_i^2\right)$ is used to prevent overfitting by penalizing the magnitude of the parameters. The pairs (u, i) for which $\mathbf{R}_{ui}$ is known are stored in the set $\mathcal{K} = \{(u, i) \mid \mathbf{R}_{ui} \text{ is known}\}$. The model parameters can be learned using optimization algorithms such as Stochastic Gradient Descent (SGD) or Adam.

In the base implementation, 64 latent factors were used with a batch size of 128, and the Adam optimizer with a learning rate of 10^{-4}. The matrix factorization model was built using the Keras deep learning framework, incorporating embedding layers for users and items to capture latent factors. It takes `user_id` and `movie_id` as inputs, computes their dot product, and optimizes the MAE using Adam.

4.3 Figure of the Model

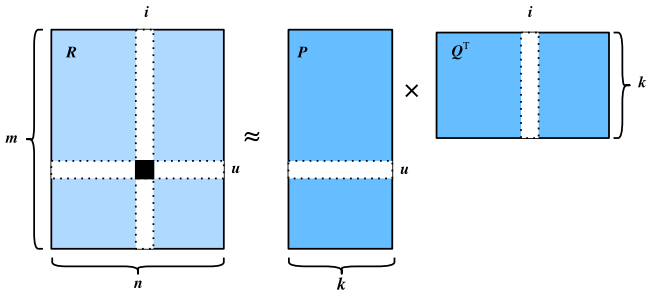

Figure 1: Matrix factorization model illustration.

4.4 Neural Network

In addition to matrix factorization, we implemented a neural network based in the matrix factorization model for the recommendation system. This model learns latent factors for users and movies through embedding layers, followed by dense layers for final rating predictions.

The architecture embeds both `user_id` and `movie_id` into latent vectors. We used 50 latent factors, with a batch size of 128, and the Adam optimizer with a learning rate of 10^{-4}. The neural network was built using the Keras framework, with embedding layers for users and movies, followed by dense layers for the rating prediction.

The key components of the architecture are:

- **Input Layer**: Accepts `user_id` and `movie_id` as inputs, each represented as a single integer.
- **Embedding Layers**: Separate embedding layers for users and movies to learn 50 latent factors.
- **Flatten Layers**: Latent factors are flattened into vectors.
- **Dropout Layers**: Dropout with a rate of 40% is applied to prevent overfitting.
- **Similarity Calculation**: Dot product between user and movie vectors captures interactions.
- **Dense Layers**: A dense layer with 96 units (ReLU activation), followed by a dense layer with a linear activation for the final rating.
- **Training**: The model is trained to minimize the Mean Absolute Error (MAE) using the Adam optimizer.

The network was trained for 10 epochs using the MovieLens dataset, with a batch size of 128. Dropout was applied after each embedding and dense layer to prevent overfitting. The model was compiled with the Adam optimizer, and MAE was used as the loss function.

5 GENETIC ALGORITHM

To optimize the hyperparameters of the recommendation models, we employed a Genetic Algorithm (GA). GAs are inspired by the process of natural selection and are widely used for solving optimization problems, particularly when the search space is large and complex. In this project, the goal was to minimize the Mean Absolute Error (MAE) by effectively exploring the hyperparameter space for both the matrix factorization and neural network models after 10 epochs.

5.1 Overview of Genetic Algorithms

A Genetic Algorithm simulates the evolutionary process by iteratively improving a population of candidate solutions through genetic operations such as selection, crossover, and mutation. Each candidate solution, also known as an individual, represents a specific set of hyperparameters for the model. The algorithm evolves these solutions over generations to find the optimal configuration.

The key steps in a GA are as follows:

- **Initialization**: A population of individuals is generated, each representing a unique set of hyperparameters (batch size, learning rate, latent factors). The initial population is randomly generated within predefined bounds.
- **Selection**: We use the tournament selection method. A group of individuals is randomly chosen from the population, and the individual with the lowest MAE is selected as a parent.
- **Elitism**: To ensure the best solutions are preserved, elitism is applied. Specifically, the top 10% of individuals (rounded to the nearest integer) with the lowest MAE are directly copied to the next generation without any modification.
- **Crossover (Recombination)**: Selected individuals undergo crossover based on a predefined crossover probability. For each hyperparameter, there is a 50% chance of inheriting the value from either parent. This process creates offspring that combine characteristics from both parents, introducing variability and enabling the exploration of new combinations of hyperparameters.
- **Mutation**: Random changes are applied to some offspring, slightly altering their hyperparameter values. This introduces diversity into the population and prevents premature convergence to local optima.
- **Evaluation and Replacement**: The new generation is evaluated, and the process continues for a set number of generations or until a stopping criterion *patience* is met.

5.2 Implementation Details

In our implementation, the Genetic Algorithm was used to optimize three key hyperparameters for both models:

- **Batch Size**: Defines the number of samples processed before the model's internal parameters are updated. The search space for this parameter was set between 64 and 512.
- **Learning Rate**: Controls how much the model's weights are adjusted with respect to the gradient. The search space was logarithmic, ranging from 10^{-10} to 10^{-2}.
- **Latent Factors**: In the matrix factorization model, this represents the number of latent features used to represent both users and items. The search space ranged from 32 to 256 latent factors.

The Genetic Algorithm was configured with the following parameters:

- **Population Size**: 50 individuals per generation.
- **Selection**: Tournament selection with a size of 2.
- **Elitism**: Enabled, preserving the top 10% of individuals (rounded to the nearest integer) for the next generation.
- **Crossover Rate**: 85% of individuals undergo crossover.

- **Mutation Rate**: 15% mutation probability per individual.
- **Generations**: The algorithm was executed for 10 generations.

6 RELATED WORK

In most cases, genetic algorithms (GA) are used in recommender systems to find optimal similarity matrices, similarity functions, or feature weights. Some of the work done by other researchers in this area is discussed in this section.

Bobadilla et al. [3] proposed an approach to calculate similarity between users using a genetic algorithm, improving collaborative filtering recommender system results and performance. They formulated a similarity function, which is a linear combination of values and weights. Values are calculated for each pair of users between whom the similarity is obtained, while weights are calculated once using a prior stage in which the genetic algorithm extracts weightings from the recommender system depending on the specific nature of the data.

Fong et al. [4] worked on a feature-based approach using a genetic algorithm for hybrid modes of collaborative filtering in online recommenders. They encoded input variables into genetic algorithm chromosomes to optimize the recommendations.

Zhang et al. [5] introduced a genetic clustering algorithm to partition source data using similarity computation techniques to address the scalability issue in collaborative filtering.

Xiao et al. [6] proposed an item-based collaborative filtering system that involved a novel similarity function using average ratings for each user instead of the overall average rating for all users.

Lastly, LV et al. [7] developed an item-based recommender system using a genetic algorithm to optimize feature weights for the recommended items.

These works demonstrate the versatility of genetic algorithms in optimizing similarity matrices and feature weights. In this paper, we focus on optimizing hyperparameters (batch size, learning rate, latent factors) for matrix factorization and neural network models using genetic algorithms to minimize the Mean Absolute Error (MAE).

7 DESCRIPTION OF THE EXPERIMENTS AND RESULTS

This section presents the experimental setup and results obtained by applying the genetic algorithm to optimize hyperparameters in two recommendation models: **Matrix Factorization (MF)** and **Matrix Factorization with Neural Network (MFNN)**. The experiments were conducted using two widely used datasets, **MovieLens 100K (ML 100K)** and **MovieLens 1M (ML 1M)**, from the GroupLens research lab at the University of Minnesota. The key hyperparameters optimized include batch size, learning rate, and latent factors. The **Mean Absolute Error (MAE)** was used as the primary performance metric.

7.1 MAE Improvement

The table below highlights the initial and optimized MAE values for each model and dataset:

Model	Dataset	Initial MAE	Optimized MAE
MF	ML 100K	2.1841	1.0329
MF	ML 1M	1.3042	1.1226
MFNN	ML 100K	0.9327	0.9342
MFNN	ML 1M	0.9645	0.8726

Table 1: Improvement in MAE after Genetic Algorithm Hyperparameter Optimization.

7.2 Optimized Hyperparameters

The next table shows the initial and optimized hyperparameters found by the genetic algorithm for each model:

Model	Dataset	Initial Params	Optimized Params
MF	ML 100K	Batch: 128 LR: 0.001 LF: 64	Batch: 226 LR: 0.0009 LF: 60
MF	ML 1M	Batch: 128 LR: 0.001 LF: 64	Batch: 246 LR: 0.0005 LF: 57
MFNN	ML 100K	Batch: 128 LR: 0.001 LF: 50	Batch: 39 LR: 0.002 LF: 78
MFNN	ML 1M	Batch: 128 LR: 0.001 LF: 50	Batch: 34 LR: 0.008 LF: 78

Table 2: Initial and Optimized Hyperparameters for each Model and Dataset.

8 CONCLUSION

The results of the genetic algorithm applied to hyperparameter optimization demonstrate significant improvements in reducing the Mean Absolute Error (MAE) for the Matrix Factorization (MF) model, particularly for the MovieLens 100K dataset. The optimized hyperparameters led to a lower MAE compared to the initial configurations, validating the genetic algorithm's effectiveness in discovering better hyperparameter sets. The substantial improvements observed across both datasets confirm the algorithm's ability to explore optimal hyperparameter spaces for matrix factorization models.

However, the results for the Matrix Factorization with Neural Network (MFNN) model were less consistent. While the genetic algorithm provided some improvements, especially for the MovieLens 1M dataset, it failed to outperform the initial hyperparameter settings for the MovieLens 100K dataset, where the initial MAE was lower than the optimized result.

Overall, the genetic algorithm proved to be effective in enhancing the MF model's performance, particularly by reducing the MAE. Although improvements in the MFNN model were more limited, the

experiment highlighted the potential for hyperparameter optimization. Future work could focus on exploring alternative optimization methods or more complex neural network architectures to achieve further performance gains in MFNN models.

9 CITATIONS AND BIBLIOGRAPHIES

REFERENCES

[1] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. In *Computer*, 42(8), 30–37.

[2] Andreas Töscher, Michael Jahrer, and Robert Bell. 2009. The BigChaos solution to the Netflix Grand Prize. Netflix Prize documentation, 1–52.

[3] J. Bobadilla, F. Ortega, A. Hernando, and J. Alcalá, "Improving collaborative filtering recommender system results and performance using genetic algorithms," *Knowledge-based systems*, vol. 24, no. 8, pp. 1310–1316, 2011.

[4] S. Fong, Y. Ho, and Y. Hang, "Using genetic algorithm for hybrid modes of collaborative filtering in online recommenders," in *2008 Eighth International Conference on Hybrid Intelligent Systems*, pp. 174–179, IEEE, 2008.

[5] F. Zhang and H.-y. Chang, "A collaborative filtering algorithm employing genetic clustering to ameliorate the scalability issue," in *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)*, pp. 331–338, IEEE, 2006.

[6] J. Xiao, M. Luo, J.-M. Chen, and J.-J. Li, "An item-based collaborative filtering system combined with genetic algorithms using rating behavior," in *International Conference on Intelligent Computing*, pp. 453–460, Springer, 2015.

[7] G. Lv, C. Hu, and S. Chen, "Research on recommender system based on ontology and genetic algorithm," *Neurocomputing*, vol. 187, pp. 92–97, 2016.