Movie Recommendation AI using KNN

Jack Shang

Computer Engineering

University of British Columbia

Often, I find myself sitting in front of a screen, in the mood for a movie, but totally unsure what to pick. I'll ask friends or family for suggestions, but most of the time, their recommendations miss the mark. It's not because I dislike their taste, it's just that I don't have a consistent preference for a particular genre or actor. My tastes are a bit all over the place. And from talking to others, I realized I'm not alone in this. A lot of people just want something *similar* to what they recently enjoyed, rather than a generic top-rated list, or some WatchMojo video. That's why I decided to build a simple movie recommendation AI. The idea is that you input a movie you liked, and it outputs five similar movies you'll probably enjoy too.

This sort of thing is already done at a large scale by platforms like Netflix, Prime Video, and YouTube. Their systems are a lot more complex, using massive datasets that track your entire viewing history, along with engagement metrics like watch time and clicks. My implementation, however, is intentionally simpler. I wanted to strip the idea down to the essentials: enter a single movie title, get five related recommendations, no user profiles, no login, no browsing history.

To build this system, I used the MovieLens Small dataset from 2018. It's a compact but rich dataset that includes movie titles, genres, and user ratings which is plenty to work with for a prototype.

Data Collection

For this project, I chose the MovieLens Small dataset (2018) as the foundation for building the recommendation system. MovieLens is a well-known benchmark dataset curated by GroupLens at the University of Minnesota, widely used in research and industry for evaluating recommender systems. I opted for the Small version because it strikes a good balance between size and usability-it contains enough data to uncover meaningful patterns in user preferences without overwhelming memory or processing constraints during development. The dataset includes around 100,000 ratings from over 600 users on approximately 9,000 movies, along with metadata such as movie titles, genres, and timestamps. This made it particularly suitable for testing a prototype model like mine, which aims to recommend movies based on user rating similarities. Moreover, the dataset is clean, well-documented, and publicly available, which made preprocessing and analysis significantly easier. Its standardized structure also ensures reproducibility and compatibility with common machine learning tools, including SciPy and pandas.

Choosing the Right Model

My first step was figuring out what kind of machine learning model would work best for the task. Since the goal is to identify and recommend movies that are similar to a movie the user already likes, I need a model that can quantify similarity between items, in this case, movies. While this isn't strictly a regression or classification problem in the traditional sense, I did initially consider using models like linear regression or logistic regression. But they didn't quite fit the nature of the problem. What I really needed was a model that can work based on distance or proximity in some feature space. That naturally led me to K-Nearest Neighbors (KNN).

Why KNN Makes Sense Here

KNN is an intuitive and non-parametric algorithm that works by measuring the "closeness" between data points in a multidimensional feature space. For classification problems, KNN predicts the class of a data point based on the majority class among its K nearest neighbors. For regression problems, it averages the values of the K closest points.

But in my case, I'm not trying to classify or predict a numerical value, I just want to *recommend* movies that are similar. So what makes KNN a good fit here is that it directly supports similarity-based lookups. Each movie in the dataset becomes a point in space, defined by features like user ratings. When a user inputs a movie, the model simply finds the K closest movies (using a similarity metric like cosine distance), and returns them as recommendations.

What makes this especially elegant is that KNN doesn't require training in the traditional sense. It doesn’t make assumptions about the underlying data distribution. Instead, it lazily evaluates similarity at query time. This is great for flexibility and interpretability, which are useful when building a prototype from scratch.

Preprocessing the Dataset

One of the bigger challenges I faced was cleaning and prepping the MovieLens dataset. Ratings are sparse, not every user has rated every movie, which makes the data hard to process directly. To get around this, I pivoted the dataset into a user-movie matrix, where each row is a movie and each column is a user rating. If a particular user hadn’t rated a certain movie, I filled that entry with a 0.0.

To prevent these “zero” ratings from introducing bias or noise, I converted the matrix into a Compressed Sparse Row (CSR) format using SciPy. This keeps memory usage efficient and ensures that unrated movies don’t skew similarity scores.

Building the Recommender

I then implemented the actual recommendation engine using the `NearestNeighbors` class from SciPy, with cosine distance as the metric and the brute-force algorithm for neighbor search. The core of the engine is a function called `recommend()`, which takes in a movie title and returns a list of five movies that are most similar based on user rating patterns.

Cosine similarity worked well here because it focuses on the angle between vectors, not their magnitude. This is useful when comparing user ratings, since two users might rate movies on different scales (e.g., one harsh and one generous), but still agree on *which* movies they like or dislike.

Results and Reflection

Evaluating a recommendation model like this isn't as straightforward as testing accuracy in classification or MSE in regression. There's no "ground truth" for what the best recommendation is. Instead, the results have to be judged qualitatively, do the recommended movies actually seem similar to the input movie?

In my tests on Google Colab, the results were promising. The system recommended movies that were either in the same genre, had similar themes, or shared user-rating patterns with the input movie. It wasn't perfect, but it definitely captured the spirit of what I was going for. In the absence of a formal accuracy metric, I relied on intuition and common sense, and based on that I'd say it worked well.

Ultimately, I'm happy with how the project turned out. It solved the problem I set out to tackle: providing movie recommendations based on a single movie you liked, without needing a complex user profile or history. While this is just a simplified proof-of-concept, it opened up my understanding of how real-world recommendation systems work, and how powerful simple models like KNN can be when used thoughtfully.