## **Comparison of Predictive Model Pathways**

| Feature / Step | Pathway A: Gradient Boosted Ranking | Pathway B: Deep Learning Feature Extraction |
| --- | --- | --- |
| **Core Idea** | Treat this as a classic, brute-force machine learning problem. Use all 380+ columns as features to directly predict a user's level of engagement with a restaurant. It's powerful, interpretable, and fast to implement. | Treat the 360 transaction and craving columns as a raw signal containing a user's "taste DNA." Use a neural network (an Autoencoder) to first learn a compressed, meaningful representation of this DNA, and then use that learned representation to make recommendations. |
| **Clustering Algorithm** | **K-Means**  • **Why**: In this context, clustering is a **feature engineering** step. K-Means is fast and simple, perfect for creating a single, powerful categorical feature (user\_cluster\_id) that summarizes user demographics and medical history. The main model can then leverage this feature.  • **How**: Apply K-Means on the user-specific columns (age, user\_lat, user\_long, and the medical condition flags). Add the resulting cluster ID back into the main DataFrame as a new column. | **HDBSCAN**  • **Why**: After using a neural network to create a rich, dense embedding for each user, you want a more powerful clustering algorithm to find the *true* underlying structure. HDBSCAN excels here because it can find clusters of complex shapes and sizes and doesn't require you to guess the number of clusters beforehand.  • **How**: First, train the Autoencoder. Then, use it to convert each user's behavioral profile into a dense vector (e.g., size 32). Finally, apply HDBSCAN on these vectors to group users by their learned taste profiles. |
| **AI/ML Model** | **LightGBM** (or XGBoost)  • **Why**: This is the industry standard for high performance on large, tabular datasets like yours. It is exceptionally fast, memory-efficient, and designed to handle hundreds of features. It will likely give you the best predictive accuracy in the shortest amount of time.  • **How**: **1.** Engineer a target variable, for example, total\_orders = the sum of all month\_X\_menu\_item\_Y columns for that row. **2.** Train the LightGBM model to predict total\_orders using all other columns (user info, restaurant info, cravings, medical flags, and the new user\_cluster\_id) as features. | **Autoencoder** (Neural Network)  • **Why**: An Autoencoder is an unsupervised neural network that learns to compress data into a small number of meaningful dimensions (an "embedding") and then reconstruct it. The compressed embedding becomes a powerful, automatically generated feature that represents a user's entire taste profile in a way that's impossible to hand-craft.  • **How**: **1.** Train the Autoencoder on the 360 transaction/craving columns. **2.** Once trained, use the "encoder" part of the network to create a dense vector (e.g., size 32) for each row. This vector *is your new feature*. **3.** You now have a simpler problem: recommending items to users based on their learned 32-dimensional "taste vector." |
| **How Recommendations are Made** | For a given user, create candidate rows pairing them with all nearby restaurants. Use the trained LightGBM model to predict the total\_orders score for each. The final recommendation is the list of restaurants, sorted by the highest predicted score. | Once users are clustered based on their "taste vectors," recommendations can be made more simply. For a user in "Cluster A," you can recommend the most popular or highest-rated restaurants among other users in "Cluster A." The heavy lifting is done in the feature learning stage. |