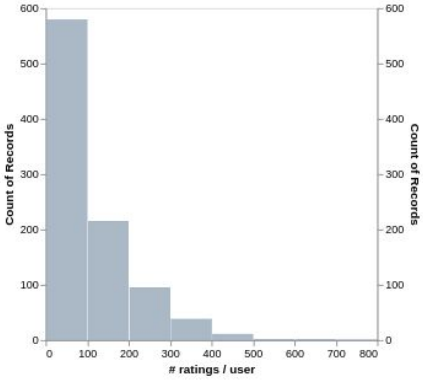
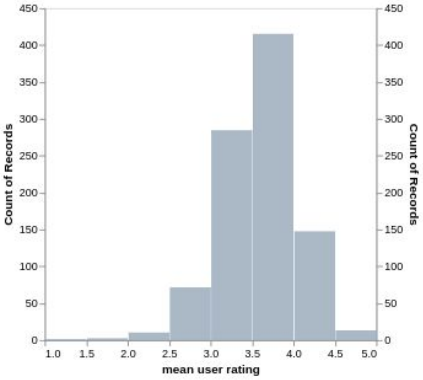
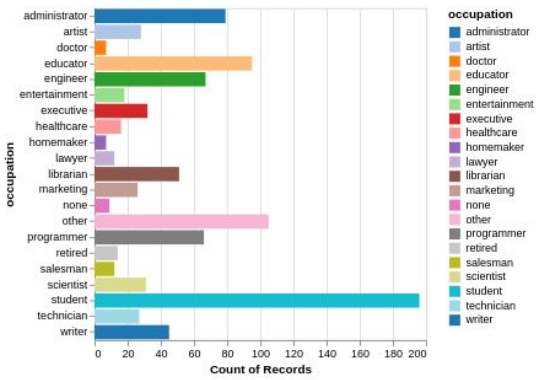
Navnit Kumar : Movie Recommendation System

EDA

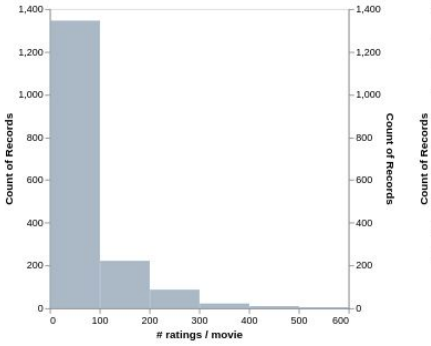
Users

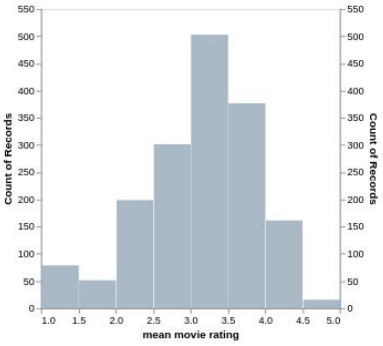


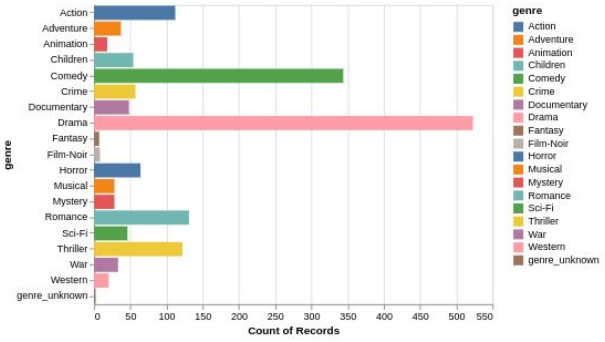




Movies







METHOD :

Our goal is to factorize the ratings matrix A into the product of a user embedding matrix U and movie embedding matrix V, such that $A \approx U V^{\top}$

- N is the number of users,
- M is the number of movies,
- A_{ij} is the rating of the jth movies by the i th user,
- each row U_i is a d-dimensional vector (embedding) representing user i
- each row V_j is a d-dimensional vector (embedding) representing movie j
- the prediction of the model for the (i, j) pair is the dot product $\langle U_i, V_j \rangle$

Calculating the error

The model approximates the ratings matrix A by a low-rank product $UV^\top$. We need a way to measure the approximation error. We'll start by using the Mean Squared Error of observed entries only (we will revisit this later). It is defined as

$$\begin{aligned}\text{MSE}(A, UV^\top) &= \frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}(A_{ij} - (UV^\top)_{ij})^2 \\ &= \frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}(A_{ij} - \langle U_i, V_j\rangle)^2\end{aligned}$$

where Ω is the set of observed ratings, and $|\Omega|$ is the cardinality of Ω.

> Note: One approach is to compute the full prediction matrix $UV^\top$, then gather the entries corresponding to the observed pairs. The memory cost of this approach is $O(NM)$. For the MovieLens dataset, this is fine, as the dense $N \times M$ matrix is small enough to fit in memory ($N = 943, M = 1682$).
>
> Another approach (given in the alternate solution below) is to only gather the embeddings of the observed pairs, then compute their dot products. The memory cost is $O(|\Omega|d)$ where d is the embedding dimension. In our case, $|\Omega| = 10^5$, and the embedding dimension is on the order of 10, so the memory cost of both methods is comparable. But when the number of users or movies is much larger, the first approach becomes infeasible.

Training a Matrix Factorization model - CF model

Matrix Factorization model using SGD

Inspecting the Embeddings

- computing the recommendations
- looking at the nearest neighbors of some movies,
- looking at the norms of the movie embeddings,
- visualizing the embedding in a projected embedding space.

Compute the scores of the candidates, given a query and item embedding

There are different similarity measures we can use, and these can yield different results. We will compare the following:

- dot product
- Cosine

We can compute recommendations, where the query embedding can be either a user embedding or a movie embedding.
It seems that the quality of learned embeddings may not be very good. This will be addressed by adding several regularization techniques.
First, we will further inspect the embeddings.

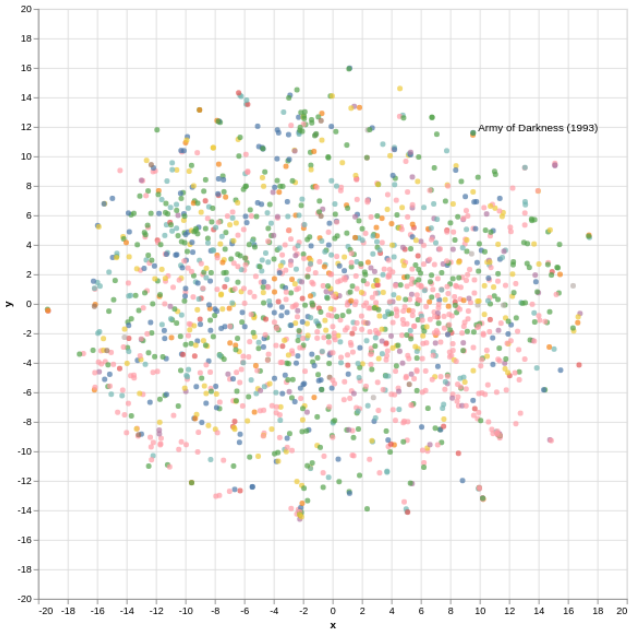
Movie Embedding Norm

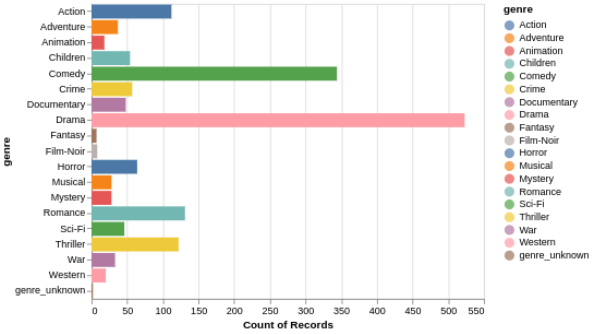
Recommendations with dot-product and cosine are different: with dot-product, the model tends to recommend popular movies.
This can be explained by the fact that in matrix factorization models, the norm of the embedding is often correlated with popularity (popular movies have a larger norm), which makes it more likely to recommend more popular items. We can confirm this hypothesis by sorting the movies by their embedding norm

Note: Depending on how the model is initialized, we may observe that some niche movies (ones with few ratings) have a high norm, leading to spurious recommendations. This can happen if the embedding of that movie happens to be initialized with a high norm. Then, because the movie has few ratings, it is infrequently updated, and can keep its high norm. This will be alleviated by using regularization.

Embedding visualization

It is hard to visualize embeddings in a higher-dimensional space (when the embedding dimension k >3), one approach is to project the embeddings to a lower dimensional space. T-SNE (T-distributed Stochastic Neighbor Embedding) is an algorithm that projects the embeddings while attempting to preserve their pairwise distances. It can be useful for visualization, but one should use it with care.





Regularization In Matrix Factorization

Our loss was defined as the mean squared error on the observed part of the rating matrix. This can be problematic as the model does not learn how to place the embeddings of irrelevant movies. This phenomenon is known as *folding*.

We will add regularization terms that will address this issue. We will use two types of regularization:

- Regularization of the model parameters. This is a common ℓ_2 regularization term on the embedding matrices, given by $r(U,V) = \frac{1}{N}\sum_i \|U_i\|^2 + \frac{1}{M}\sum_j \|V_j\|^2$.
- A global prior that pushes the prediction of any pair towards zero, called the *gravity* term. This is given by $g(U,V) = \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}\langle U_i, V_j\rangle^2$.

The total loss is then given by

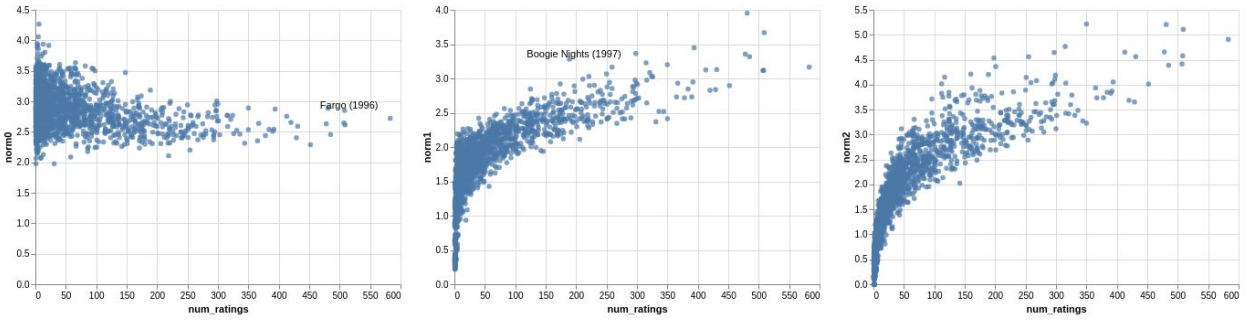
$$\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}(A_{ij} - \langle U_i, V_j\rangle)^2 + \lambda_r r(U,V) + \lambda_g g(U,V)$$

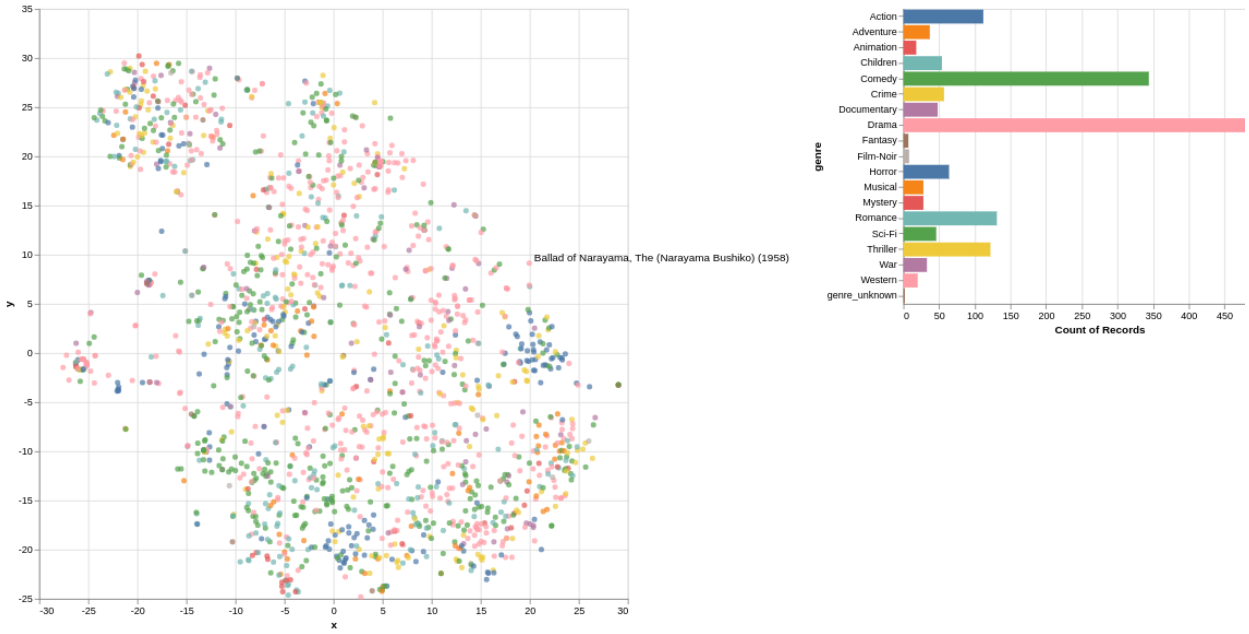
where λ_r and λ_g are two regularization coefficients (hyper-parameters).

Build a regularized Matrix Factorization model, use gravity loss and train it

Adding the regularization terms results in a **higher MSE**, both on the training and test set. However, as we will see, the quality of the recommendations improves.

This highlights a tension between fitting the observed data and minimizing the regularization terms. Fitting the observed data often emphasizes **learning high similarity** (between items with many interactions), but a good embedding representation also requires **learning low similarity** (between items with few or no interactions).





Observe that the embeddings have a lot more structure than the unregularized case.

Note that while the scale of the problem is small enough to allow efficient training using SGD, many practical problems need to be trained using more specialized algorithms such as Alternating Least Squares.

Softmax model

We will train a simple softmax model that predicts whether a given user has rated a movie. The model will take as input a feature vector x representing the list of movies the user has rated. We start from the ratings DataFrame, which we group by user_id.

Loss function

Recall that the softmax model maps the input features x to a user embedding $\psi(x) \in \mathbb{R}^d$, where d is the embedding dimension. This vector is then multiplied by a movie embedding matrix $V \in \mathbb{R}^{m \times d}$ (where m is the number of movies), and the final output of the model is the softmax of the product

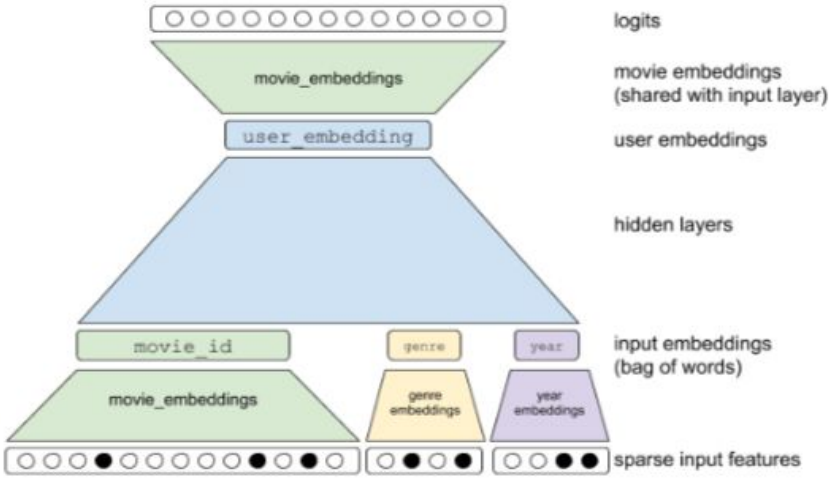
$$\hat{p}(x) = \text{softmax}(\psi(x)V^\top).$$

Given a target label y, if we denote by $p = 1_y$ a one-hot encoding of this target label, then the loss is the cross-entropy between $\hat{p}(x)$ and p.

Function takes tensors representing the user embeddings $\psi(x)$, movie embeddings V, target label y, and return the cross-entropy loss.

Build a softmax model, train it, and inspect its embeddings

softmax CFModel



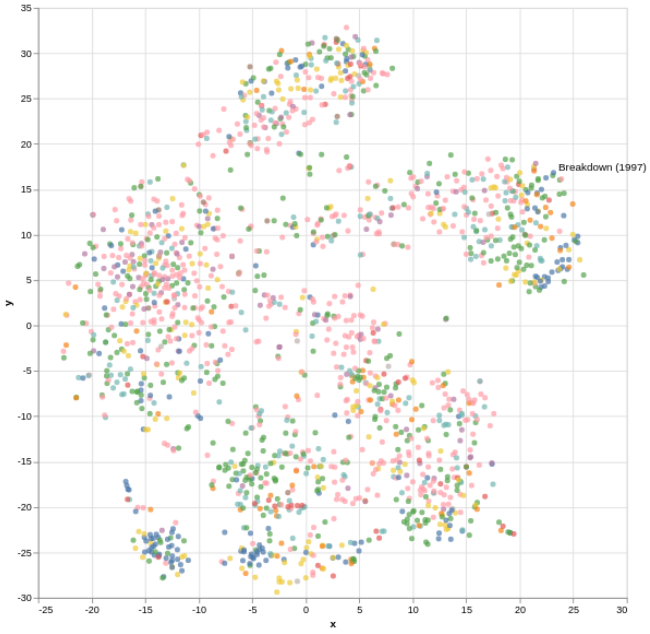
Train the Softmax model

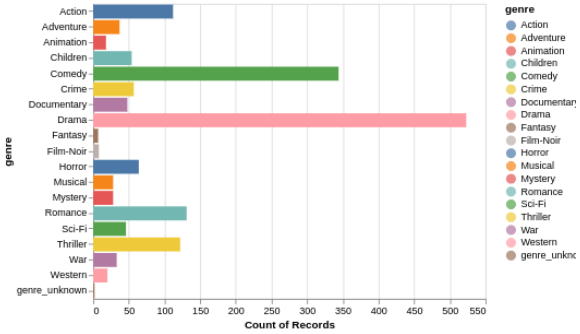
We can set the following hyperparameters:

- learning rate
- number of iterations. Note: we can run softmax_model.train() again to continue training the model from its current state.
- input embedding dimensions (the input_dims argument)
- number of hidden layers and size of each layer (the hidden_dims argument)

Inspect the embeddings

Note that in this case, the movie embeddings are used at the same time as input embeddings (for the bag of words representation of the user history), and as softmax weights.





We can further explore these models and are encouraged to try different hyperparameters and observe how this affects the quality of the model and the structure of the embedding space. Some suggestions:

- Change the embedding dimension.
- In the softmax model: change the number of hidden layers, and the input features. For example, try a model with no hidden layers, and only the movie ids as inputs.
- Using other similarity measures: we used dot product and cosine and discussed how the norms of the embeddings affect the recommendations. We can also try other variants which apply a transformation to the norm.