Recommender Systems

Movielens 100k Dataset

Karran Pandey

Collaborative Filtering and Matrix Decomposition along with Dimensionality reduction techniques to predict User movie ratings

About the dataset:

Movielens datasets were collected by the GroupLens Research Project at the University of Minnesota.

This data set consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies.
- Each user has rated at least 20 movies.
- Simple demographic info for the users (age, gender, occupation, zip)

Data description:

One sample data point:

Consists of the format UserId, MovieId, Rating, Timelapse.

For e.g - 196 242 3 881250949

Methodology:

We used 6 different recommender methods to predict user ratings across the data sets.

Namely we set up Collaborative Filtering along with its baseline variant and Matrix decomposition techniques like Singular Value Decomposition and CUR. Moreover, we also implemented dimensionality reduction in these matrix decomposition techniques which preserved 90% of the energy so as to increase the computational efficiency while keeping the accuracy similar for these methods.

Pre-processing:

- Initially the data was in the format of [User ID, Movie ID, Rating, Timelapse]
- In order to apply any of the recommender systems we operate on a User – Movie Rating matrix, where each row represents a user, and each column – a movie, with the elements of the matrix containing the ratings
- Using the pandas dataframe and numpy the matrix was converted to the required User – Movie format.

Collaborative Filtering:

We implemented User-User collaborative filtering. As per the following formula:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Where s represents the cosine similarity between the two ratings.

The size taken for neighbouring region is assumed to be 5: that is we used the 5 most similar users to estimate our rating. We enforce another condition that we use only those users who have actually rated the movie as our sample space for these operations, so as to get only useful information.

The baseline approach was implemented using the following formula:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$b_{xi} = \mu + b_x + b_i$

- μ = overall mean movie rating
- b_x = rating deviation of user x = (*avg. rating of user x*) – μ
- b_i = rating deviation of movie i

Singular Value Decomposition:

SVD was done for the whole processed data matrix(let A) using the eigen value decomposition of the matrix. Numpy's eigh function was used to find the normalized eigen vector matrices for AA^T and A^TA, which were then flipped from left to right in order to compute U and V respectively.
It was also assumed that, the first element of each eigen vector is positive and the vectors were made uniform in a directional sense using the assumption.

Sigma is a diagonal matrix made from the eigen values of AA^T and A^TA.

Results were compared with pythons inbuilt numpy.linalg.svd function for correctness. Our svd function returned the same results.

CUR Decomposition:

CUR decomposition was done by sampling 100 columns and 100 rows from the data matrix based on a probability distribution formulated as:

$P(x) = \sum A(i,x)^2 / \sum A(i,j)^2$ (for columns)

$P(x) = \sum A(x,j)^2 / \sum A(i,j)^2$ (for rows)

Where we take sum over each row i and column j. Using np.random.choice(array,prob_dist_array), we could sample rows and columns based on the given probability distributions, to get C and R.

We then went on to find the SVD of the intersection matrix for C and R, and take its Moore Penrose pseudo inverse, given by:

$$U = Y(\Sigma^{+})^2 X^{T}.$$

For a matrix who's SVD consists of X, sigma and Y. Finally, U is the required U as per CUR decomposition.

Dimensionality Reduction based on energy:

For dimensionality reduction we took as input, three matrices U,sigma and V, representing a decomposition. We then proceeded to remove a column and row at a time from the central diagonal matrix, until the energy was the lowest possible value atleast 90% of the original energy.

Energy is formally defined as the sum of the squares of the diagonal elements of the sigma matrix.

To do the dimensionality reduction with each removal of a row and column from sigma, we removed the last row of V and the last column of U.

After completing this process, we continued the same removal process for all three matrices to remove from sigma if there existed any zero rows or zero columns, till sigma was a square matrix.

Predicting ratings using matrix decompositions:

To predict user ratings from a matrix decomposition U,sigma,V we first made a concept matrix where we mapped all the users to the concept space through multiplication with V transpose.

We then mapped the user in question to the concept space and found the most similar user in the concept space who had rated the movie. We then assigned this most similar user's rating as our prediction for the user in question's rating.

Statistics:

All statistics were calculated on 1000 sized randomly selected test data, for which true value was known, for prediction.

Reduced SVD gave 271 x 271 matrix of sigma from 943 x 943

Running time for all is on 100 predictions, for SVD and CUR it's inclusive of the time to form the matrix. Similarly for dim. Reduction case it's only inclusive of time to reduce matrix.

Technique Used	RMSe	Precision at top 100	Spearmans Correlation	Running time
Collaborative Filtering	1.4261258227	0.33	0.999987796979	46s
Collaborative Filtering with baseline	1.44490186672	0.35	0.999987473539	40s
SVD	1.65106026541	0.39	0.999983643984	70s
SVD (with dim. Reduction)	1.68047612301	0.33	0.999983055983	30s
CUR	1.60779351908	0.4	0.999984489984	5s
CUR(with dim. Reduction)	1.66132477258	0.34	0.999983439983	6s

Data Structures Used:

A pandas dataframe was used initially to help process the data easily into the required format.

The processed data was in the form of a numpy 2D array.

From then on as everything was in vector or matrix form, nothing outside of numpy arrays were used as data structures to represent the actual form of the variables, as vectors or matrices.