Chase Robertson
CS 401R
11 April 2018

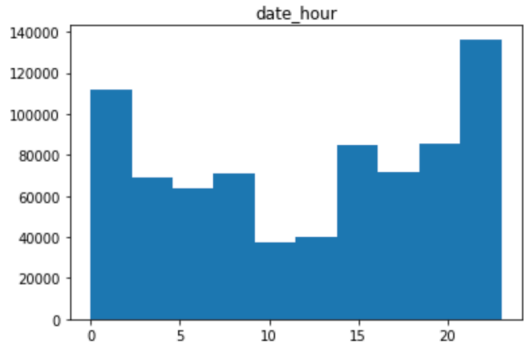
Recommender System

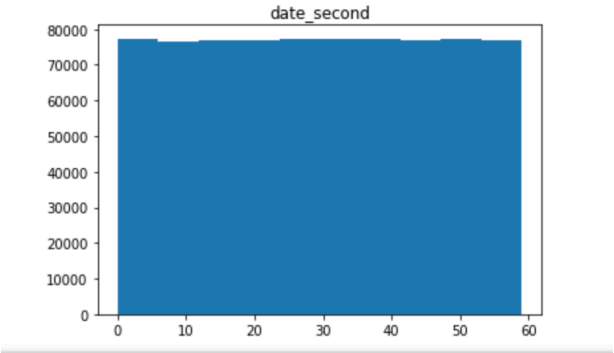
Part 1: The Data

The first thing I did after importing the movie ratings data was print out the shape of the data table to get a feel for the scale. I then printed the first few data instances to see what features were included and what the actual data values looked like. Next, I printed a histogram of each feature to get a feel for the distribution of all instances. I noticed that a few features had nearly flat histograms, indicative of a uniform distribution over that feature. This made sense when I looked closer at which features showed this uniformity: instanceID, date_minute, and date_second. Each of these features should not have any substantial differences or effect on any given user's rating of any given movie, so it made sense that they showed a uniform distribution.

I also visualized the auxiliary data file information with histograms to get a feel for how that data was distributed but did not end up using any of it for reasons of simplicity.

The following graphs illustrate the difference between the distribution of a potentially informative feature: 'date_hour', and a mostly useless one: 'date_second'.





My first exploration into making predictions was to set each test instance prediction to the mean movie rating in the training set. This provided me with a baseline RMSE accuracy of 1.0 with which to judge my later approaches.

Part 2: The Model

My first step in training a model to make movie rating predictions was to partition my dataset into training and validation sets. I randomly chose 85,000 instances, roughly 10% of the data, to act as a validation set for my predictions.

My approach included filling in missing values of the user/movie matrix represented as its SVD, using stochastic gradient descent. This approach has a rich history of use in predicting movie ratings, and for many other sparse matrix ML problems.

My model uses 1/3 the number of features in the training data as the number of factors in the SVD, in order to best fit the available data. With roughly 6 billion possible user/movie ratings, the SVD enabled me to reduce matrix entries to only a million or so, while still modeling the 6 billion entry matrix closely. The SVD is composed of 2 matrices, the product of which is something close to the actual user/movie rating matrix. The user matrix has all possible users by the number of factors, where the movie matrix has all possible movies by the number of factors. The dot product of any given user's factor vector with a movie's factor vector should result in an accurate prediction of that user's preference, whether or not that user has already rated that movie.

I initialized the factor vectors of each SVD matrix with uniformly distributed random values between 0 and 0.1. Training this model included iteratively updating the factor vectors of the corresponding user and movie according to the rating given in each training instance. This update depended on a learning rate and regularization constant to vary the degree to which the vector could be updated between iterations and epochs of training. I elected not to include any bias, as it added complexity that slowed my training more than it added to prediction accuracy.

Part 3: The Performance

My final validation set RMSE was 0.8137

I know I didn't overfit on the training set because my validation set was kept separate throughout the process and I stopped training before validation RMSE bottomed out. My first algorithm included gradient descent with bias but was not the one I ultimately used for submission. My impatience with the time it took to execute eventually convinced me to reduce the complexity of the inner loop, and the most obvious complexity to remove was bias. I found that prediction accuracy was not adversely affected by this change, so I stuck with it.