HW4: Movie Recommender System

Bhavika Tekwani
btekwani@gmu.edu

Introduction

In this assignment, we are tasked with building a recommender system making the best use of the ratings and additional data. The challenges faced when building a recommender system are sparse data, feature selection (in the case of additional user and item data) and user bias (users have their own patterns of rating an item high or low).

Rank and RMSE score

As of 11/13/2016, 8:15 PM, my rank was 1 and RMSE score was 0.76.

Team Name

bolt79

Approach

Feature selection and engineering

We have been provided with files containing genre information, actors and directors for each movie. Additionally, there are also tags assigned to the movies by users.

My approach was to initially use only the training data to set a benchmark for RMSE. Then I incrementally tried several combinations of genre, actors, directors and movie tags to observe the variation in RMSE. In recommender systems, two types of data are frequently used to generate recommendations. One is user data i.e information about the user which may be demographic information (age, gender, favourite items, social activity) and the other is item information which in the case of our movie recommender system is genre, actors and directors.

When we think about movies, we know that the cast greatly influences our decision to watch the movie or engage with it using tags on social media. Genre is another important feature - it is often observed that some users watch more of action movies than say, romantic movies. Some users watch more documentaries than comedies. Our dataset contains a lot of rich features but combining all of them introduces the curse of dimensionality.

I filtered actors based on the number of movies they appear in and set a threshold (no of appearances as 20). It may be argued that a good actor appears in a select number of very highly rated movies (3.5+) than a mediocre actor who appears in a large number of movies with less than average ratings (<3.4). Using the same logic for directors, I selected 20 directors who occur mostly frequently in the dataset.
I used one hot encoding to generate a `DataFrame` that consisted of just the 19 actors, 20 directors and all genres for each movie. I used this as my final feature set for the solution I've submitted.
The reduction in features looks like this - there are 46187 unique actors, which were reduced to 19. 4060 directors were reduced to 20. Genre data was used as it is, although it can be pruned to remove less popular genres like Western (261 movies), Film Noir (145), IMAX (25) and Short (1). The effect of removing these features can be explored further. This feature set gave me a training RMSE of 0.53 using cross validation on a 50-50 distribution of the train and test data. This method of feature pruning was driven by running cross-validation multiple times - a more aggressive threshold for the number of actors and directors was leading to overfitting (RMSE was as small as 0.3 on the training set but 0.8+ on the leaderboard).

Prediction

I'm using a library called GraphLab to perform matrix factorization. Our problem of predicting a rating between 1 and 5 for a user, movie combination is called an 'explicit data' problem. GraphLab provides a simple interface to tune parameters and factorize the main user-item matrix. I used the `FactorizationRecommender` provided by GraphLab to factorize the train data and side data (different combinations of genres, actors, directors and tags). The internal coefficients of the model are learned from known scores of users and items. Recommendations are then based on these scores. `FactorizationRecommender` works by learning two types of coefficients - factor terms which are between users and movies and latent factors which capture the global relationship between all the side data for movies and users. If a user tends to like action movies more than romantic ones, the factor terms would capture that, thereby giving lower ratings to romantic movies and higher to action movies.
When no side data is used, the predicted score for user i based on item j is given by,

$$\text{score}(i,j) = \mu + w_i + w_j + a^T x_i + b^T y_j + u^T{}_i v_j$$

Where μ is the global bias term, w_i is the weight for user i, w_j is the weight for item j, x_i and y_i are the side feature vectors for user i and item j, a and b are weights for those feature vectors and u_i and v_j are the latent factors (same length as number of factors the matrix is factorized to).

Using a library like `pyFM` for matrix factorization limited me to a RMSE of 0.94 on the leaderboard. I used `FactorizationRecommender` to involve the side features and boost the performance once I was clear on my methodology. `FactorizationRecommender` is a matrix factorization machine which approximates target rating values as a weighted combination of user and item latent factors, biases, side features, and their pairwise combinations. Under the hood, this model can use different solvers like Stochastic Gradient Descent (SGD), Alternating Least Squares (ALS) and Adaptive Gradient Descent (Adagrad). I used Adagrad because it's a gradient optimizer well suited for sparse data. Adagrad also eliminates the need for manual tuning of the learning rate.

Improvements

These are the approaches I would try to get a better RMSE score:

1. Define a popularity metric that correlates the presence of certain actors and directors in a movie with the rating.
2. Prune the tags data based on weight, vectorize them and use those vectors as additional features for the movie data
3. Prune the training set to remove users that occur infrequently and movies that have not been rated by many users (“rare” movies) and train on this reduced dataset.

References

[1] Ruder, Sebastian. "An Overview of Gradient Descent Optimization Algorithms." Sebastian Ruder. N.p., 16 Sept. 2016. Web. 12 Nov. 2016. <http://sebastianruder.com/optimizing-gradient-descent/index.html#adagrad>.

[2] Fang, Yi, and Luo Si. "Matrix Co-factorization for Recommendation with Rich Side Information and Implicit Feedback." Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems - HetRec '11 (2011): n. pag. Web. 12 Nov. 2016. <http://www.cse.scu.edu/~yfang/hetrec11-fang.pdf>.

[3] Rendle, Steffen. "Factorization Machines." 2010 IEEE International Conference on Data Mining(2010): n. pag. Web. 12 Nov. 2016. <http://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf>.