Crystal Paudyal, Stefan Himmel, Kailash Pandey
CS 0451 Final Project Write Up
December 7, 2017

Recommendations and Chill

Briefly summarize the technique/problem/application that you investigated:

In this project we tried to tackle the Netflix Prize contest from 2006 to develop the best collaborative filtering algorithm to recommend Netflix users movies related to the ones they have rated. We used the basic collaborative filtering algorithm described by Andrew Ng in his Coursera course to get our model, and then used a Pearson's R algorithm to predict the movies for a given user (looks at the predicted ratings of all the movies that a user has not yet watched, and then recommends the movies with the highest predicted rating).

Your experimental setup. What data did you use? How did you set up the experiment (10-fold cross-validation, etc.)? What did you use for evaluation? How did you decide if results were significant?

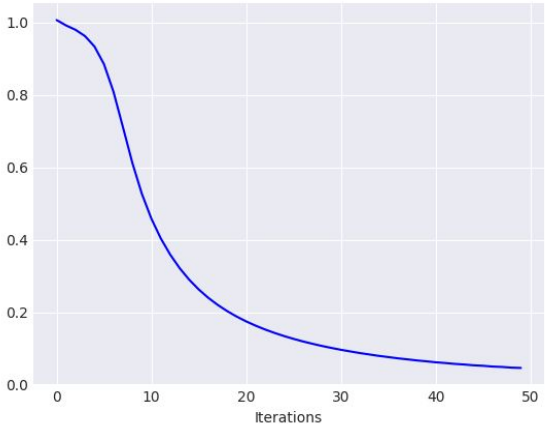
For data, we used the 640+ MB data set of Netflix's 100,480,507 ratings that 480,189 users gave to 17,770 movies. We began by parsing this dataset, cutting out movies that received too few ratings / users that rated too few movies, and then created a matrix with movies as rows, users as columns, and ratings as values. When we started, we realized that if we were to create the entire (movies x users) matrix, it would contain over 8.5 billion values. Because there were only around 100 million actual ratings, our matrix would be mostly empty, so we began looking into ways to store only the needed ratings to conserve memory and computation time. We decided to try using a sparse data-structure called a data frame (similar to a matrix) by Pandas. This seemed to be working, but we ran into an enormous amount of trouble with how these sparse matrices behaved doing simple arithmetic. We eventually decided to just switch back to Numpy matrices and use Gattaca. Ultimately, we implemented random initialization of a θ matrix for movies and a X (feature) matrix for users, normalized our Y (movies x users) matrix, wrote functions to calculate Costs/Gradients, performed gradient descent, and then calculated calculated the final RMSE (root mean squared error) as our evaluation metric, and finally made recommendations with Pearson's R. We also compared the lowest Cost/RMSE we could get with our gradient descent algorithm with the results generated from passing our cost function and gradients into a highly optimised cost-reduction function called fmincg.

Your results. These should be stated concisely and should include supporting tables, graphs and figures.

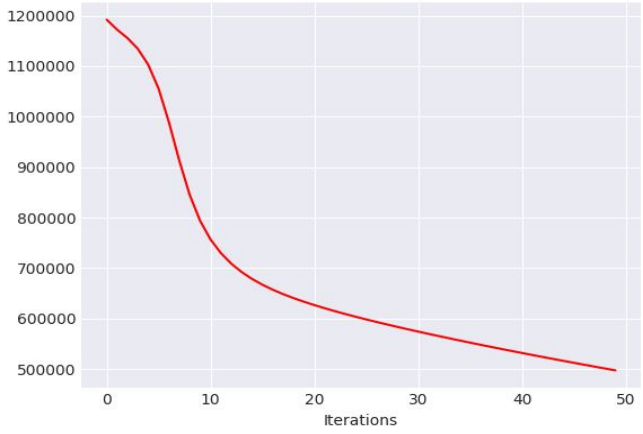
During testing our algorithm we:

1. Made sure that our Cost and RMSE calculated on the difference between our y-hat and y in a small dataset of the first 220 movies were both going down.

(Cost)



(RMSE)



2. Tuned our hyperparameters on this little dataset to get the lowest cost/RMSE we could:

Alpha = 0.0025
Lambda = 1.5
Epochs = 50
Number of Features = 1000

**ideally we would have done this on the full dataset, but the time/memory constraints made it implausible for this project*

Our gradient descent algorithm:

- Lowest cost: 497385
- Lowest RMSE: 0.04605

For fmincg:

- Lowest cost: 19057
- Lowest RMSE: 0.03714

3. Made movie recommendations for user x all high predictions of movies user y has not rated, and made sure that they were all highly rated predictions and that x's recommendations were different for user y. For example:

Top recommendations for user 132:
Predicting rating 4.7 for movie Invader Zim.
Predicting rating 4.6 for movie That '70s Show: Season 1.
Predicting rating 4.5 for movie Silkwood.
Predicting rating 4.5 for movie Lord of the Rings: The Return of the King: Extended Edition: Bonus Material.
Predicting rating 4.4 for movie Ninotchka.
Predicting rating 4.4 for movie Fame.
Predicting rating 4.3 for movie Husbands and Wives.
Predicting rating 4.2 for movie A Little Princess.
Predicting rating 4.2 for movie Elfen Lied.
Predicting rating 4.2 for movie Richard Pryor: Live on the Sunset Strip."

4. Finally, we ran the algorithm on the entire dataset, then calculated (for the movies and users specified in the "probe.txt" dataset: see here for more details https://www.netflixprize.com/faq.html#probe) the RMSE between yhat and y to compare to the evaluation standard that Netflix gave for the original competition.

Netflix's Cinematch: algorithm got a 0.9474 RMSE on the probe data set

We got: 0.053283 (using "big220.txt", or the ratings on the first 220 movies to train, and then testing RMSE for all of the user/movie pairs in the "probe.txt" that were in the first 220)

*We were ultimately unable to test our algorithm on the entire dataset, as Gattaca ultimately gave us a memory error. We suspect that our RMSE would be much larger if we were able to calculate it on the full dataset.

As a fun fact, BellKor's Pragmatic Chaos team (winners) got 0.8567 RMSE on a test set that is no longer available (can't see what they got on the probe dataset either).

Conclusions. Summarize your findings.

Overall, our implementation was somewhat successful. We were able to create our own collaborative filtering algorithm that was (somewhat) comparable to a more optimized version (fmincg). Unfortunately, we are not able to fully evaluate our algorithm, because we could not train it on the entire dataset due to memory constraints, and could thus not properly compare our RMSE to the Netflix standard. If we were to go forward with the project, we would simply need to return to our original approach of working with a sparse matrix (or pandas dataframe),

and rewrite our cost/gradient descent functions to accommodate that, as everything else (including the probe testing) is in place. For the time-frame we were working with, we are happy with the outcome and have all learned a lot about collaborative filtering, and the overall process of and difficulties in machine learning.

CODE: https://drive.google.com/open?id=15nv9pgXJqH89MzhgvYc_H-fgL52bKZfl