CSCI 181 / E-181 Spring 2014 Practical 2

David Wihl
davidwihl@gmail.com

Zack Hendlin
zgh@mit.edu

February 24, 2014

Warm-Up

Maximum Likelihood Estimation

Max likelihood

Bayesian Linear

Bayesian linear

Locally Weighted Linear Regression

Locally weighted linear regression

Recommender System

For the main part of the exercise, I investigated a series of increasing complex algorithms.

Pearson Distance

The first was using Pearson distance from *Programming Collective Intelligence*.[1]

$$r = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}}{\sqrt{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}}\sqrt{\sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}}} \tag{1}$$

Unfortunately Pearson distance is not very effective with sparse data. Given that the training consisted of only 200000 ratings for 131378 books $\times$ 12787 users, the ratings were sparse, so Pearson was not very effective at all.

[1] Programming Collective Intelligence by Toby Segaran. © 2007 Toby Segaran, 978-0-596-52932-1.

Collaborative Filtering with Regularized Gradient

In the previous Coursera course, I had to build a similar recommender system for movies. (The vectorized Octave implementation can be found in `cofiCostFunc.m`.) However that data was significantly different with 1682 movies and 943 critics. The biggest difference was again the sparseness of this problem's data in comparison.

An additional complexity was translating the existing Octave code to Python. While I had significant Python programming experience, I was less familiar with the `numpy` and `scipy` vector libraries. The Coursera / Ng class neatly packages the implementation so there were only a few lines of vectorized Octave to add. To reimplement in Python was rapidly taking significantly more code (like 20x!) to get equivalent functionality.

While I could have adapted the prior Octave code to this problem, I decided not to as 1) I would not have learned as much and 2) I was skeptical that the algorithm would adapt well to the sparseness of the current data set. In my real world experience, data is sparser and noisier so it would be more interesting to tackle a new approach.

A More Systematic Approach

Given the limit of four Kaggle submissions per day, I could not simply attempt a wide variety of different methods. While Kaggle would provide an overall score quality, it would not provide enough fine-grained feedback as to which cases were lowering the score. Also, each execution was taking up to 20 minutes on my laptop. So I chose a more systematic approach. First, I created a very simple set of training data. This synthetic training data allowed me to exercise a variety of different permutations and edge cases.

Normal cases:

- two very similar users
- two very similar books
- two or more mostly similar users

Pathological cases:

- an outlier user (e.g. single review)
- a book without any reviews
- a user without any reviews

This is more of a Test-Driven Development approach where most cases, both plausible and implausible, are defined prior to implementation. *The most limiting factor in this exercise is the number of Kaggle submissions, not the data nor the algorithms.* I needed a means to improve the quality of the implementation quickly without using up a Kaggle submission unnecessarily.

To further improve the quality, I split the 200000 rows of training data into an 80/20 mixture of training and validation data. The validation set enabled me to check that the code would apply readily to the problem data as well verify the accuracy of the predictions. It also enabled me to find which patterns of data had the most error. Different algorithms could be applied depending on the pattern, a kind of simplified ensemble system.

Cosine Distance

The next algorithm I attempted with a rather simple Cosine Distance[2] calculation.

$$\cos(x, y) = \frac{x \cdot y}{|x| \times |y|} \tag{2}$$

Since Cosine distance works well with sparse data, it significantly improved the results. This also exercised the synthetic data and testing environment. However, I was still below the per-user mean, so the implementation had to increase in sophistication.

Researching the Netflix Solutions

I read and reviewed the winning Netflix solutions, specifically, Feature-Weighted Linear Stacking[3] and Matrix Factorization Techniques[4]. However, as a solo Extension school student without prior `numpy` experience, I felt I did not have sufficient time to build a reliable implementation of those more sophisticated algorithms.

Interestingly, Netflix ended up using only parts of the Progress Prize stage (which already represented 2000 hours of work and a combination of 107 different algorithms). They never used the final $1M winning solution due to the additional engineering complexity for a limited improvement in rating prediction quality. More importantly, they shifted their business from predicted ratings to broader and more effective means of recommendations[5].

Conclusion

This exercise raised some important questions about overfitting. Fairly generic algorithms for recommendations that would work well with a different data set would fail with this data set. I would imagine that even the best of class solution for this data set would fail miserably with an Amazon or Netflix-sized set of data in terms of in-memory matrix realities, more sparseness of data (millions of products across hundreds of categories for example) and higher dimensionality of person and item features (buying history, multiple people-per-household, evolution of ratings over time). I would further research how to

[2]Guide to Data Mining, Chapter 2 http://guidetodatamining.com/chapter-2/

[3]Sill, et al. http://arxiv.org/pdf/0911.0460.pdf

[4]Volinsky et al. http://www2.research.att.com/ volinsky/papers/ieeecomputer.pdf

[5]http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

determine quickly the limits and appropriateness of a given algorithm to a given data set. Is it even possible to have reasonably generic learning algorithms at all? If the implementation needs significant re-writes for just the rating prediction class of problem, how will we tackle the great variety of potential machine learning applications?

I really enjoyed this exercise. The Kaggle competition was a strong motivating factor to improve my results, especially as the leaderboard varied day-by-day or hour-by-hour. The real world sparse and noisy data was a more effective learning exercise than the packaged Octave example in my prior Coursera Machine Learning class. The need to use a systematic approach with both clean and pathological test data will be useful both in future exercises for this class as well as real world application.

As an Extension student, I would have liked to have the opportunity to partner with either other Extension students or in-class students. I think this would have improved the learning rate for both my potential collaborators and me. In the Coursera class, I often learned as much from other students as from the lectures and class material. Of course, I also reciprocated with my practical experience in Machine Learning applications from my day job. Education is becoming a blend of in-class and on-line learning. Like the Kaggle competition, mixing the two would be a more modern, more effective approach. Ensemble learning for the classroom!