CSCI E-181 Spring 2014 Practical 1

David Wihl
davidwihl@gmail.com

February 12, 2014

Warm-Up

As a warmup, I synthesized five clusters of data. I then used a K-Means implementation in Octave I had written for a previous course.[1] While this implementation was sufficient for the prior course's provided dataset, when I tested it the synthesized data set, K=5 and random initial centroids, one of the centroids would frequently not converge on any points.

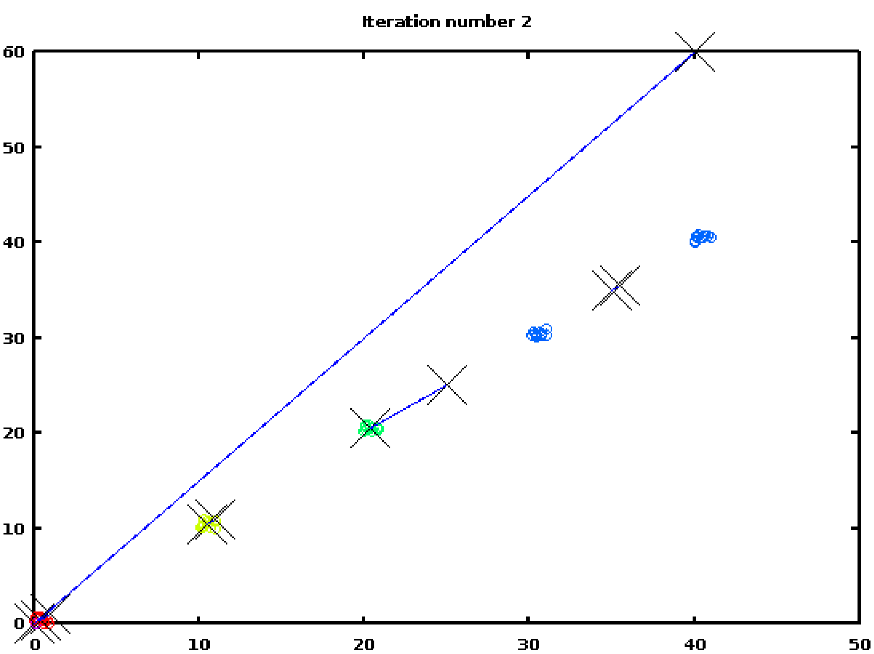

Figure 1: Random Initial Centroids After 1 Iteration

[1]Machine Learning, Coursera, Prof. Andrew Ng, Completed Jan 2014, https://class.coursera.org/ml-004

I subsequently modified the code to use K-Medoids, choosing one of the sample data points at random as an initial centroid. This worked much better.

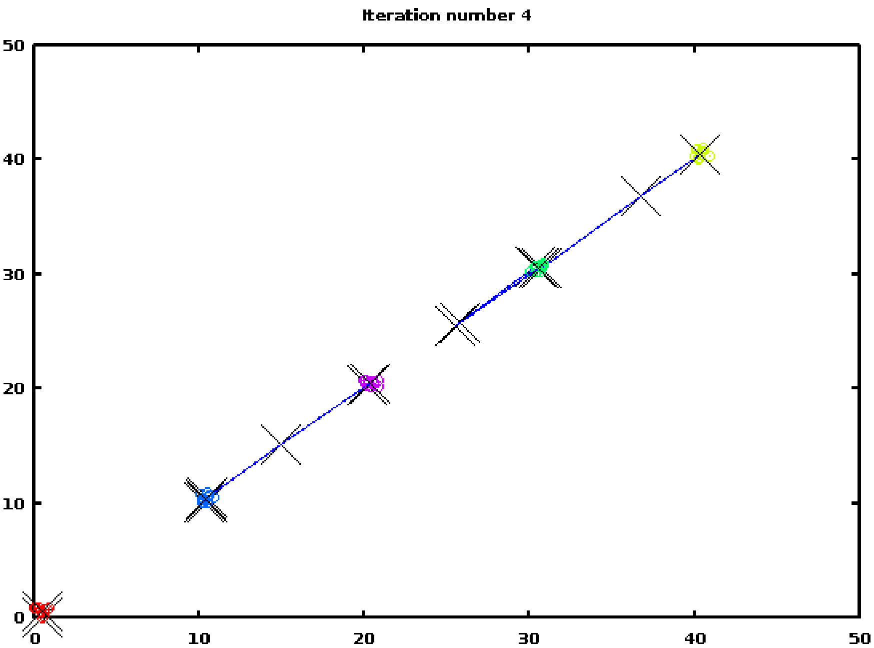

Figure 2: K-Medoids Converge After 4 Iterations

CIFAR-10 Image Data

I then attempted using K-Medoids with the CIFAR-10 Image Data, using the Matlab version of the data with Octave. The training data consists of a 10000x3072 matrix of UInt8. Each row is a 32x32x3 (total 3072 columns) color image, consisting of 1024 red, 1024 green and 1024 blue elements. There are 10 classes in the set ("airplane", "automobile", etc.), so setting K=10 was a rational first step.

Percentage Distribution of K values after normalization and 10 iterations

06 05 04 26 14 13 05 04 03 15

TODO: fill this in

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*.[2]

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

Figure 3: Pearson Correlation Coefficient Approximation

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

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 `./warmup/cofiCostFunc.m`.) However that data was significantly differently with 1682 movies and 943 critics. The biggest difference was again the sparseness of this problem's data in comparison.

$$\frac{\partial J}{\partial x_k^{(i)}} = \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta_k^{(j)} + \lambda x_k^{(i)}$$

$$\frac{\partial J}{\partial \theta_k^{(j)}} = \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda \theta_k^{(j)}.$$

Figure 4: Regularized Gradient Cost Function

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 do the equivalent in Python was rapidly taking 20x lines of Python 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, 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 more sparse and more noisy so it would be more interesting to tackle a new approach.

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

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 did 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.

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 before using up a Kaggle submission.