Predicting user ratings with incremental SVD

Stephen Merity **J. Benjamin Cook**

Abstract

Hello, World. This is an abstract.

1 Introduction

In this section, we give an introduction to our incremental SVD paper.

2 Data

The core of the Netflix dataset consists of 17,770 text files. Each text file represents a distinct movie. The first line in the text file is the movie's unique ID number, which is an integer from 1 to 17,770. All other lines have three comma-delimited entries: user ID, rating, and date.

There are 480,189 unique users in the dataset, however, IDs range from 1 to 2,649,429, with gaps. Ratings are integers from one to five indicating the number of stars the user gave to the movie in question. Dates are in the YYYY-MM-DD format, although we do not use this information in the current project. For example:

```
1:
1488844,3,2005-09-06
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
...
```

The above data come from the first movie's text file, which contains ratings for the movie *Dinosaur Planet*. The data indicate that user 1488844 gave *Dinosaur Planet* a three star rating on September 6, 2005, user 822109 gave the movie a five star rating on May 13, 2005, and so on.

In order to be able to perform SVD, we need a matrix with users on the rows and movies on the columns. This matrix would be $480,179 \times 17,770 = 8.5$ billion entries. In a regular matrix format, this would too big to hold in memory. One estimate is that it takes roughly 65 Gb of RAM to hold the entire matrix (**?**) although the actual size would depend on the size of the amount of space allocated for each rating. Fortunately, the matrix is extremely sparse, containing only around 100 million non-zero entries. The data structure we use is Python's `scipy.sparse.lil_matrix`. We store data from the text files in this sparse matrix as we read them. After reading in all of the text files, we output the matrix to a Matrix Market format. The Matrix Market format starts with a line containing the dimensions of the matrix and the number of non-zero entries. Then, each line contains $i \quad j \quad <$ value $>$. For example, these are the first few lines of a Matrix Market file with a subset of the Netflix data :

```
20000 1000 564726
1 1 3
1 8 4
1 17 2
```

```
1 30 3
...
```

Finally, because the process of implementing our incremental SVD system was iterative and because even the iterative method requires serious computational power, we reduced our dataset to smaller subsets for testing. We ran our algorithm on datasets of size 1000×2000, 1000×3000, ...

3 Method

In the context of recommender systems, the main task of Singular Value Decomposition (SVD) is to decrease the dimensionality of the dataset. We need to be able to summarize the key characteristics of a movie in a much smaller number of features, or variables, than the total number of users in the system. Similarly, we need to reduce the key preferences of users to something much smaller than the total number of movies in the database. The reasons for decreasing the dimensionality of the dataset are two-fold. First, without dimensionality reduction, machine learning tasks would be computationally intractable and second, reducing the dimensionality actually allows us to predict ratings more effectively, since our dataset is so sparse.

3.1 SVD

As with other matrix factorization techniques, SVD works by decomposing a matrix, $\mathbf{A} \in \mathbb{R}^{m \times n}$, where $m \geq n$ and $\text{rank}(\mathbf{A}) = r$, into separate matrices whose product is $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$$

Here, $\mathbf{U} \in \mathbb{R}^{m \times r}$, is composed of the eigenvectors of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$, or left-singular vectors of $\mathbf{A}$, $\mathbf{S} \in \mathbb{R}^{r \times r}$ is a diagonal matrix whose elements are the r singular values of $\mathbf{A}$, and $\mathbf{V}^{\mathrm{T}} \in \mathbb{R}^{r \times n}$ is composed of the eigenvectors of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, or right-singular vectors of $\mathbf{A}$ **?**. Furthermore, the rows and columns of $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}^{\mathrm{T}}$ are sorted in such a way that the largest singular values occur in the upper left most corner of $\mathbf{S}$. This means that we can achieve a low-rank approximation to $\mathbf{A}$ by considering taking only the k first singular values.

$$\begin{aligned}
\mathbf{A} &\approx \mathbf{A}_k \\
&= \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^{\mathrm{T}} \\
&= \mathbf{U}[:, 1:k]\mathbf{S}[1:k, 1:k]\mathbf{V}[:, 1:k]^{\mathrm{T}}
\end{aligned}$$

This approximation $\mathbf{A}_k$ is the rank k matrix that minimizes the Frobenius norm: $\|\mathbf{A} - \mathbf{A}_k\|_{\mathrm{F}}$.

Although we attempted to implement the full SVD algorithm by following the recipe laid out in **?**, it quickly became apparent that implementing SVD from scratch was beyond the scope of this project

3.2 Folding-in

Unfortunately, the SVD algorithm requires $O(m^3)$ time complexity. On the Netflix training dataset with 480,179 users, this is on the order of 10^{17} operations, which is infeasible without a big super computer. Fortunately, a technique called folding-in allows us to compute SVD on a subset of users (or movies) and then add users (movies) incrementally.

This approximation of $\mathbf{A}$ becomes worse as we fold-in more and more users, and we lose orthonormality; however, the approximation performs well enough to predict user ratings to within a tolerable amount of error. Additionally, each user can be folded-in in $O(1)$ time, meaning it is suddenly possible to approximate SVD of the whole matrix on a laptop.

Assuming we want to fold-in users, we have two parameters, the number of singular values to use, k, and the number of users to begin with, u. The procedure works as follow **?**, :

1. Compute full SVD for the first u users:

$$\text{SVD}(\mathbf{A}[1:u, :]) = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$$

2. Take the first k singular values:

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^{\mathrm{T}}$$

3. For i in $u+1$ to m:

$$c = \mathbf{A}[i,:]$$
$$c' = c\mathbf{V}_k\mathbf{S}_k^{-1}$$
Append c' to the bottom of $\mathbf{U}_k$

In order to fold-in movies instead of users, we replace the parameter u with v, the number of movies to begin with. Then step one becomes:

$$\mathrm{SVD}(\mathbf{A}[:, 1:v]) = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$$

And for step three, we repeat the following for j in $v+1$ to n:

$$p = \mathbf{A}[:,j]$$
$$p' = p^{\mathrm{T}}\mathbf{U}_k\mathbf{S}_k^{-1}$$
Append p' to the right side of $\mathbf{V}_k^{\mathrm{T}}$

3.3 Predicting ratings

After folding-in all users and movies, it is possible to use $\mathbf{U}_k$, $\mathbf{S}_k$, and $\mathbf{V}_k^{\mathrm{T}}$ to predict user ratings that we have not observed, i.e. empty elements in our ratings matrix. For example, to estimate the rating that user i would give movie j, we simply take:

$$P_{i,j} = \bar{r}_i + \mathbf{U}\sqrt{\mathbf{S}_k}^{\mathrm{T}}[i,:] \cdot \sqrt{\mathbf{S}_k}\mathbf{V}_k^{\mathrm{T}}[:,j]$$

where $\bar{r}_i$ is the row mean of the non-zero elements of $\mathbf{A}$ **?**.

4 Results

We evaluate incremental SVD in three ways. First, we hold out a test set of ratings and use our recommender system to predict what they should be. Let the vector of user ratings in the test set be y, one indication of the performance of incremental SVD is relative error, $\frac{\|y-\hat{y}\|_2}{\|y\|_2}$, where $\hat{y}$ is the vector of predictions for the ratings in the test set. Next, we asses the 'accuracy' of our approximation of $\mathbf{A}$, $\|\mathbf{A} - \mathbf{A}_k\|_{\mathrm{F}}$ where $\mathbf{A}_k$ is constructed from $\mathbf{U}_k$ that is built with folding-in. Finally, in full SVD, the columns of $\mathbf{U}$ are orthonormal, meaning $\mathbf{U}\mathbf{U}^{\mathrm{T}} = \mathbb{I}_m$. We assess the deviation from orthonormality by computing $\|\mathbf{U}\mathbf{U}^{\mathrm{T}} - \mathbb{I}_m\|_{\mathrm{F}}$.

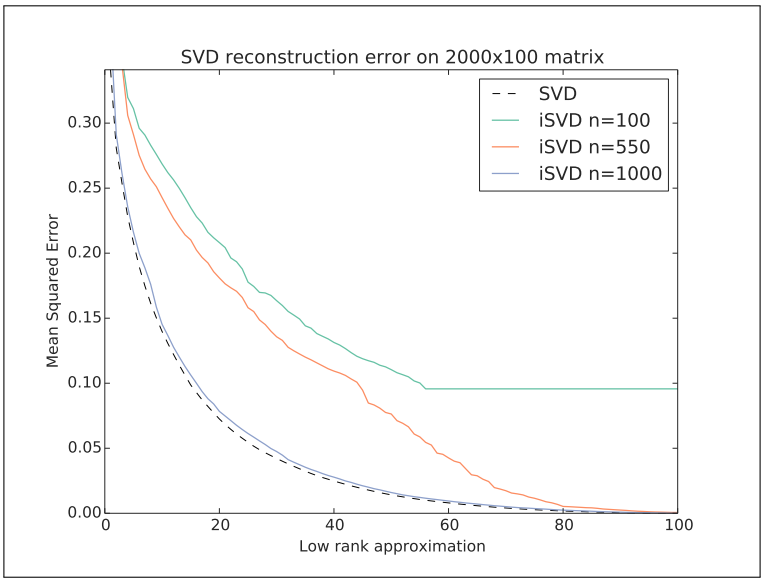

Figure 1: This is a caption

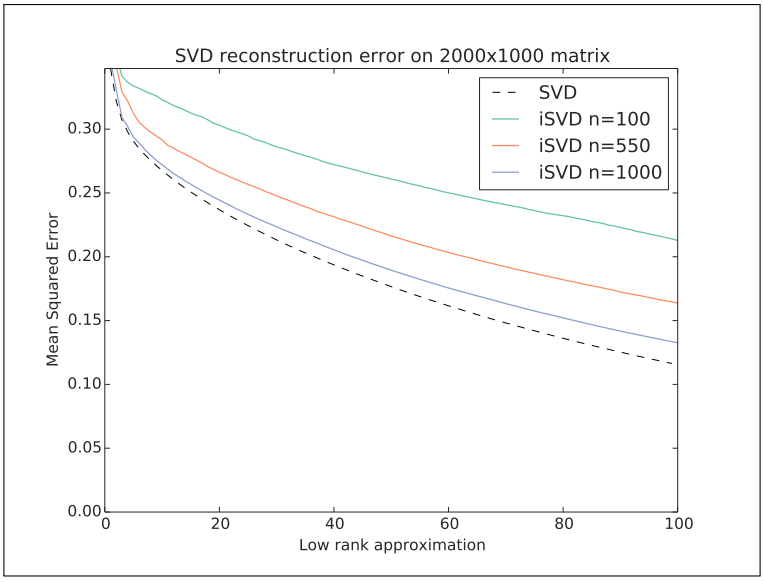

Figure 2: This is a caption

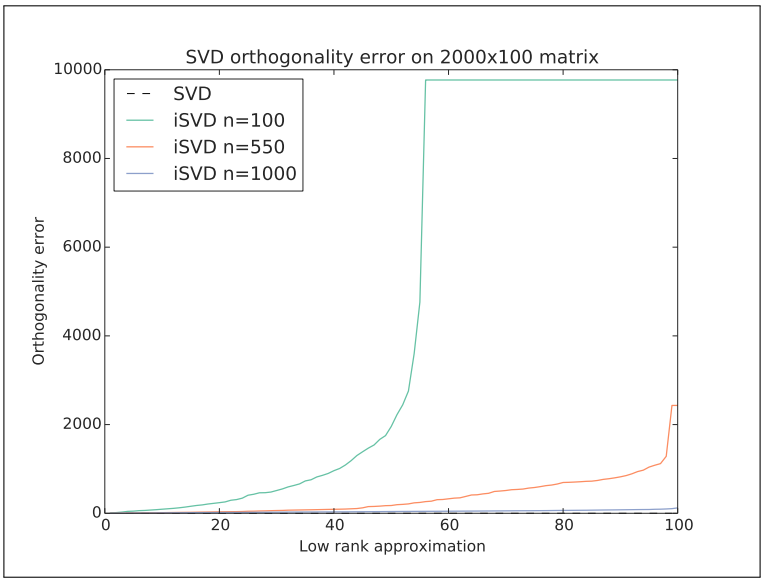

Figure 3: This is a caption

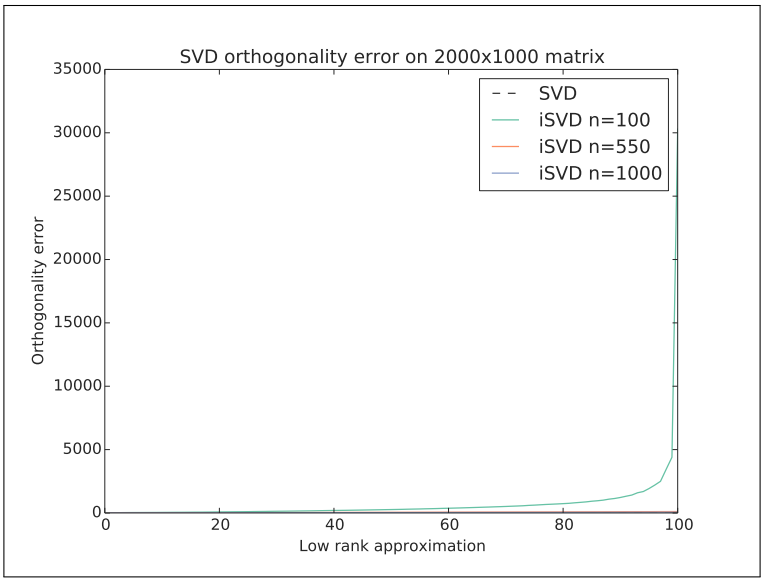

Figure 4: This is a caption

5 Discussion

This is a discussion of how well we did.

6 Conclusion

In conlusion, this paper was awesome. Cheers.