Incremental SVD for large-scale recommendation algorithms

Stephen Merity **J. Benjamin Cook**

Abstract

The singular value decomposition (SVD) is fundamental to many machine learning algorithms, primarily being used for dimensionality reduction. Unfortunately, SVD algorithms typically have quadratic complexity and require random access to the complete data set, making them impractical for many real world tasks. We investigate the convergence properties of incremental singular value decomposition (SVD) which does not require the complete data set to be stored in memory. We then compare this with traditional SVD algorithms to see the impact on memory usage, speed of convergence, and orthogonality of the resulting SVD. To motivate incremental SVD in the context of machine learning, we construct a recommendation system and show the performance of SVD compared to incremental SVD, showing the potential to incrementally build SVD-based models and produce a highly scalable recommendation system.

1 Introduction

Recommendation systems aim to recommend an item to interested potential customers, tailoring the choice of item based upon both that customer's history and the history of the all users. The items recommended can be divserse, including movies, music, web pages, or other general products. These systems have evolved from novelties to vital tools that are re-shaping the world of e-commerce. By learning from both the customer and the broader community as to which items should be recommended to someone, these successful recommendations can lead to substantial improvements in both revenue and customer satisfaction (Schafer et al., 1999).

Within recommendation systems, there are two primary types: *content-based* recommenders and *collaborative filtering* recommenders. Content-based approaches analyze the content, such as the text, metadata, or features of an item, to identify related items. A known successful realization of content-based recommendation is music recommendation on services such as Pandora or Spotify. Hundreds of features are created for each song, either manually or automatically. These features aim to capture the significant characteristics of the piece of music, allowing recommendations for a user either explicitly through their own preferences (the user stating they like jazz) or implicitly through the user's past behaviour.

An alternative approach, which is content agnostic, is collaborative filtering. Collaborative filtering analyzes the relationships between user choices within a given community to make recommendations. A major appeal of this method is that it is domain free and does not need any information about the item being recommended. Though it does suffer from the *cold start* problem[1], given large amounts of user history, collaborative filtering is the most popular method for recommendation systems. Collaborative filtering is widely deployed across the Internet, seeing specific popularity after the introduction of the Netflix prize competition in 2006. FIX

Explain matrix factorization methods in terms of collaborative filtering.

[1]The *cold start* problem refers to the issue of making recommendations before a large enough history of user behaviour is available.

In this paper, we give an introduction to collaborative filtering on large datasets using incremental SVD.

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, with their IDs ranging 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 (Smith, 2011) although the actual size would depend on the amount of space allocated for each rating. Fortunately, the matrix is extremely sparse, containing around 100 million non-zero entries. To store the data in our project, we use SciPy's `scipy.sparse.lil_matrix` which constructs sparse matrices using row-based linked lists. 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, $\ldots$. FIX

3 Method

In the context of recommendation 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. 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 $\mathbf{A}$:

$$\mathbf{A} = \mathbf{USV}^{\mathrm{T}}$$

Here, $\mathbf{U} \in \mathbb{R}^{m \times r}$, is composed of the eigenvectors of $\mathbf{AA}^{\mathrm{T}}$, 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 $\mathbf{A}$ (Golub and Reinsch, 1970). 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 Press (2007), it quickly became apparent that implementing SVD from scratch was beyond the scope of this project. It would be unlikely that we could use our SVD implementation on the full dataset either FIX

3.2 Folding-in

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 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. In the context of recommendation systems, however, the loss of orthonormality is not necessarily a core concern. The approximation produced by the incremental SVD algorithm performs well enough to predict user ratings to within a tolerable amount of error, especially as the exact initial values are already unknown. 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 described in Antulov-Fantulin (2011) states that to compute the incremental SVD of a matrix:

1. Compute full SVD for the first u users:
$$\text{SVD}(\mathbf{A}[1:u, :]) = \mathbf{USV}^{\mathrm{T}}$$
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:

$$\text{SVD}(\mathbf{A}[:, 1:v]) = \mathbf{USV}^{\mathrm{T}}$$

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

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

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 $\mathbf{A}$ (Sarwar et al., 2002).

4 Results

We evaluate incremental SVD in three ways. First, we hold out a test set of ratings and use our recommendation 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 assess 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 shows root mean squared error for our prediction of user ratings on the test set as k increases. We ran our prediction beginning with several different numbers of users u, displayed in the legend. Note that $u = 3,000$ is the same as SVD on the entire ratings matrix.

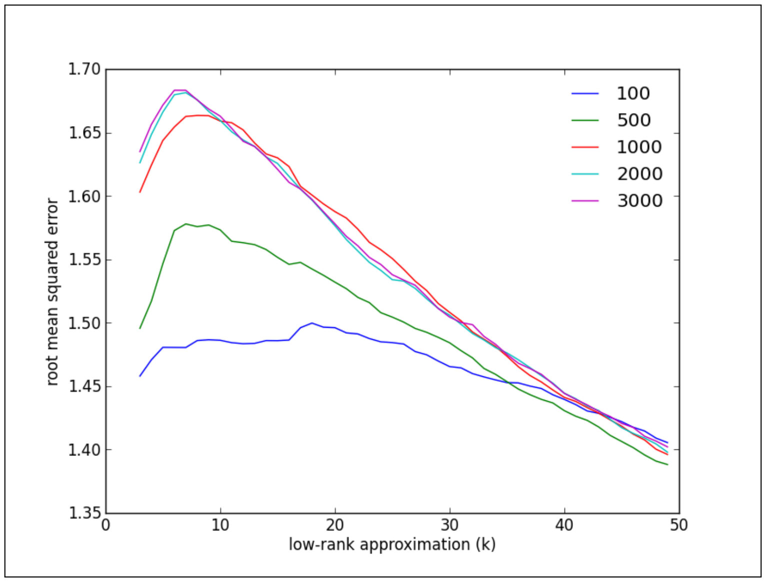

Figure 1: This is a caption

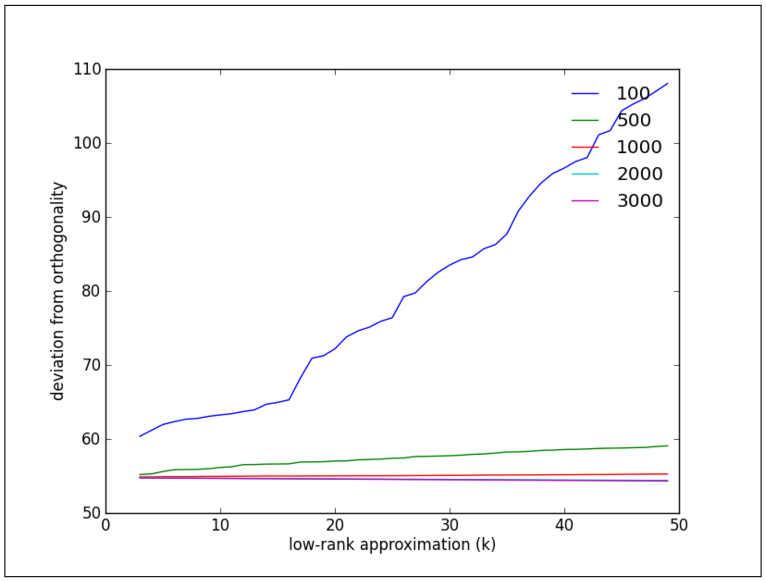

Figure 2: This is a caption

5 Discussion

This is a discussion of how well we did.

6 Conclusion

In conclusion, this paper was awesome. Cheers.

References

Nino Antulov-Fantulin. Application of Dimensionality Reduction (SVD) in Collaborative Filtering, 2011.

Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, 1970.

William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge University Press, 2007.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28. Citeseer, 2002.

J Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

David Smith. The Netflix Prize, Big Data, SVD and R. http://blog.revolutionanalytics.com/2011/05/the-neflix-prize-big-data-svd-and-r.html, 2011.