Data Analysis Using Matrix Decomposition

Tristan Gaeta

April 26, 2021

Abstract

Matrix decomposition has had a significant impact on the realm of data science. We will be discussing some important aspects of the Singular Value Decomposition (SVD), and applications of the SVD in data analysis. To demonstrate some of these applications, we will be talking about the 2006 Netflix Prize Competition to create a better recommending system, and how some participants addressed the challenge. Similar matrix decomposition techniques are in recommendation or search algorithms used by all sorts of advertisers, streaming services, and other tech companies. There are a variety of different applications of the topics we will discuss.

Key Aspects of the SVD

The traditional Singular Value Decomposition consists of the product three matrices, U, S, and V, where U and V are unitary matrices, and S is an $m \times n$ matrix with singular values along the diagonal. The SVD can

reveal correlations within data sets, and is often used for data compression. The decomposition of a matrix, X, is as follows

$$X = USV^*$$

Suppose we have a dense, $m \times n$, data matrix, X, with complex entries. The columns of X are composed of the m-dimensional vectors $\boldsymbol{x}_i$ for $1 \leq i \leq n$. Let's look into how we can interpret the correlation matrix X^*X. We know that by the definition of matrix multiplication that the entry of row i and column j of X^*X can be expressed as the following.

$$\begin{aligned}
[X^*X]_{ij} &= \sum_{k=1}^{m} [X^*]_{ik}[X]_{kj} \\
&= \sum_{k=1}^{m} \overline{[X]}_{ki}[X]_{kj} \\
&= \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle
\end{aligned}$$

In other words each entry of the matrix, $[X^*X]_{ij}$, is the Hermitian inner product of the ith and the jth columns of the original data matrix, X. If we are trying to evaluate how the vectors, $\boldsymbol{x}_i$, are related to each other, these values will be of interest. These inner products contain information about how similar any two vectors are to each other. Vectors that differ more greatly will be more orthogonal to each other, and therefore will have an inner product closer to zero. In the case of the Singular Value Decomposition, the columns of X^*X are spanned by the orthonormal basis made from the column space of the unitary matrix, U.

Similarly, the matrix XX^* is composed of the Hermitian inner products of the rows of the data matrix, X. In the Singular Value Decomposition, columns of XX^* are in the column space of the matrix, V.

The matrix, S, from the SVD captures another aspect of these correlation matrices. Let μ be the minimum of m and n. Matrices XX^* and X^*X will share the same eigenvalues, $\{\lambda_1, \lambda_2, \cdots, \lambda_\mu\}$. The remaining $|m - n|$ eigenvalues of the larger matrix will be zero. We define the singular values, σ_i, of the original data matrix, X, as $\sigma_i = \sqrt{\lambda_i}$ for $1 \leq i \leq \mu$. Singular values are sorted in decreasing order such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\mu$. Using these singular values, we define the $m \times n$ matrix, S, entrywise as

$$[S]_{ij} = \begin{cases} \sigma_i & \text{for} \quad i = j \\ 0 & \text{for} \quad i \neq j \end{cases}$$

The sorted order of singular values in matrix S is an important aspect of the SVD. Let vector $\boldsymbol{u}_i$ be the ith column of U and vector $\boldsymbol{v}_i$ be the ith column of V. Since matrices U and V are unitary, vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ will have a norm of one. Therefore, the outer product of these vectors, $\boldsymbol{u}_i\boldsymbol{v}_i^*$, will be a matrix with a Frobenius norm of one. We can think of the matrix product, USV^*, as a summation of these norm one matrices scaled by the corresponding singular value. Because of the sorted order of singular values, we will see a similar order in the Frobenius norms of the outer products, such that

$$\|\sigma_1\boldsymbol{u}_1\boldsymbol{v}_1^*\|_{\mathrm{F}} \geq \|\sigma_2\boldsymbol{u}_2\boldsymbol{v}_2^*\|_{\mathrm{F}} \geq \cdots \geq \|\sigma_\mu\boldsymbol{u}_\mu\boldsymbol{v}_\mu^*\|_{\mathrm{F}}$$

This gives rise to an important application of the SVD. We can approximate the original matrix, X, using only the larger singular values. This topic, and its applications, will be discussed further in the following section.

Matrix Approximation and Data Compression

In data analysis, it is often the case that we are working with an extremely large number of data points, which can make analysis difficult. Data sets with high dimensionality can be extremely difficult to work with for a number of reasons. If, for example, we want to use our data to solve a system of linear equations, $A\boldsymbol{x} = \boldsymbol{b}$, an exact solution only exists if the vector $\boldsymbol{b}$ is in the column space of A. Therefore if our data has high dimensionality, we will need a large number of linearly independent measurements to construct any vector $\boldsymbol{b}$. However, with a large enough data set, we would be able to approximate the vector $\boldsymbol{b}$ with a linear combination of the vectors we have. If our data contains a number of dimensions with little variance, and their exact differences are of little contribution the desired vector, they can be excluded from our approximation. Doing so compresses our data by decreasing the number basis vectors requires to approximate $\boldsymbol{b}$.

There are a several ways to decrease the number of values we need to work with for the Singular Value Decomposition. Suppose we want to perform an analysis on a dense, $m \times n$, data matrix, X, by computing its SVD. Assume that $m > n$, though there is a similar argument for the opposite case. Using the traditional decomposition, where matrices U and V are unitary, we would need $m^2 + n^2 + n$ values to represent matrices U, V, and S. We want to decrease the number of values required to represent the SVD while preserving its exactness. We know,

$$
\begin{aligned}
X = USV^* &= \begin{bmatrix} \boldsymbol{u}_1 | & \boldsymbol{u}_2 | & \cdots | & \boldsymbol{u}_m \end{bmatrix} \left[\begin{array}{cccc} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \\ \hline \multicolumn{4}{c}{\mathcal{O}} \end{array}\right] \begin{bmatrix} \boldsymbol{v}_1^* \\ \boldsymbol{v}_2^* \\ \vdots \\ \boldsymbol{v}_n^* \end{bmatrix} \\
&= \begin{bmatrix} \sigma_1\boldsymbol{u}_1 | & \sigma_2\boldsymbol{u}_2 | & \cdots | & \sigma_n\boldsymbol{u}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^* \\ \boldsymbol{v}_2^* \\ \vdots \\ \boldsymbol{v}_n^* \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{u}_1 | & \boldsymbol{u}_2 | & \cdots | & \boldsymbol{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^* \\ \boldsymbol{v}_2^* \\ \vdots \\ \boldsymbol{v}_n^* \end{bmatrix} \\
&= \widetilde{U}\widetilde{S}V^*
\end{aligned}
$$

This method is often referred to as the Economy SVD. With this method we can see that $\widetilde{U}$ is an $m \times n$ matrix made from the first n columns of U. The matrix $\widetilde{S}$ is now a square, diagonal matrix with singular values in decreasing order along the diagonal. To represent these as matrices we now only need $mn + n^2 + n$ values. (Note, similarly in the case where $m < n$, we can construct the matrix product $U\widetilde{S}\widetilde{V}^*$, where $\widetilde{S}$ is an $m \times m$ diagonal matrix and $\widetilde{V}$ is an $n \times m$ matrix from the first m columns of V). It is important to notice that if the Economy SVD is performed, the matrices $\widetilde{U}$ and $\widetilde{V}$ are no longer unitary. The matrices will still span the column space and row space of the data matrix, X, respectively, but will not form orthonormal bases for $\mathbb{C}^m$ or $\mathbb{C}^n$.

In a similar manner, we can decrease the size of our data by making an approximation of the data matrix, X. Because singular values along the diagonal of matrix S are in decreasing order, we can create a rank k approximation, $\widehat{X}_k$, that only include the k largest singular values, such that

$$
\widehat{X}_k = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^* \qquad\qquad 1 \leq k \leq \min(m, n)
$$

Suppose we want to find the $\widehat{X}_k$ closest X by minimizing the Frobenius norm, $\|X - \widehat{X}_k\|_{\mathrm{F}}$, while having the same rank, r, as matrix X. In other words, we want to solve the system

$$\min_{\widehat{X}_k}(\|X - \widehat{X}_k\|_{\mathrm{F}}) \qquad\qquad \mathrm{rank}(\widehat{X}_k) = \mathrm{rank}(X)$$

According to the Eckard-Young Theorem [7], the solution to this system would be the rank r approximation, $\widehat{X}_r$, where

$$\widehat{X}_r = \widehat{U}\widehat{S}\widehat{V}^* = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*$$

In this case $\widehat{U}$ and $\widehat{V}$ are both $m \times r$ matrices made of the first r columns of U and V respectively, and $\widehat{S}$ is an $r \times r$ diagonal matrix. This allows for quite a bit of data compression with little to no loss of accuracy. We can represent this approximation with $mr + nr + r^2$ values. When we are dealing with extremely large data sets with high demensionality this can save us a great deal of computation and memory.

The Netflix Prize and SVD

In 2006 the movie rental service Netflix, before creating their online streaming service, hosted a competition where, for the prize of one million dollars, contestants were asked to improve Netflix's current recommender system, Cinematch, by 10% or more [8]. The participants were given a data set that contained the sparse ratings from 480,189 user accounts on 17,770 different movie tittles. The set only contained 100,480,507 ratings total, each also include the date the rating was given [5].

Let's think about this problem from the perspective of matrix decomposition. We want to use our data to find correlations among users and movies to predict what person j might rate item i. Lets construct a data matrix, X, such that each column, $\boldsymbol{x}_j$, contains all of the ratings for one movie in the data set, in some order. We can assume there exists a SVD for our data matrix [6]. So if we would like to see what person i would rate item j we can

see by the definition of matrix multiplication that

$$\begin{aligned}[X]_{ij} &= \sum_{k=1}^{m} [US]_{ik}[V^*]_{kj} \\ &= \sum_{k=1}^{m} [US]_{ik}\overline{[V]}_{jk} \\ &= \langle \widehat{\boldsymbol{v}}_j, \widehat{\boldsymbol{u}}_i \rangle \qquad \widehat{\boldsymbol{v}}_j \in \mathcal{R}(V),\ \widehat{\boldsymbol{u}}_i \in \mathcal{R}(U)\end{aligned}$$

In other words we can express any user-item preference, $[X]_{ij}$, as an inner product of two vectors, $\widehat{\boldsymbol{v}}_j$ and $\widehat{\boldsymbol{u}}_i$, from the row spaces of V and U. Using the traditional SVD, these vectors would be of dimension n. With the Netflix data set, they would be vectors from $\mathbb{R}^{480,189}$, which may require a considerable amount of computation to approximate.

Unfortunately, in the case of the Netflix data set, we can't simply compute this matrix decomposition. Because the data matrix, X, has missing values, we have to approximate the vectors, $\widehat{\boldsymbol{v}}_j$ and $\widehat{\boldsymbol{u}}_i$. The best solution will minimize the error from our known data values and the magnitude of the approximations we make. Let k_{ij} be the known rating from user i of item j from the set of all known ratings, K. Let λ be a regularization factor determined by cross-validation. A good approximation will satisfy the following minimization problem [2].

$$\min_{\widehat{\boldsymbol{v}},\widehat{\boldsymbol{u}}} \sum_{k_{ij} \in K} (k_{ij} - \langle \widehat{\boldsymbol{v}}_j, \widehat{\boldsymbol{u}}_i \rangle)^2 + \lambda(\|\widehat{\boldsymbol{v}}_j\|^2 + \|\widehat{\boldsymbol{u}}_j\|^2)$$

Because every unknown $\widehat{\boldsymbol{u}}_i$ and $\widehat{\boldsymbol{v}}_j$ pair is undefined, this system is not convex. However, there are several methods we can use to approximate a solution.

Finding a Solution

Perhaps the simplest method towards approximating a minimal solution is Stochastic Gradient Descent (SGD), popularized by Simon Funk as an strategy in the Netflix prize competition [4]. This is a recursive process of calculating the error of our prediction on a known rating, and adjusting the vectors, $\widehat{\boldsymbol{u}}_i$ and $\widehat{\boldsymbol{v}}_j$, in the opposite direction of the gradient. Let's define the error, r_{ij}, of our approximation for rating k_{ij} in the set K as

$$r_{ij} = k_{ij} - \langle \widehat{\boldsymbol{v}}_j, \widehat{\boldsymbol{u}}_i \rangle$$

The vectors $\widehat{\boldsymbol{v}}_j$ and $\widehat{\boldsymbol{u}}_i$ are then adjusted by an offset proportional to γ, a data-dependent constant controlling the magnitude of our adjustment. The following represents the reassignment of vectors $\widehat{\boldsymbol{u}}_i$ and $\widehat{\boldsymbol{v}}_j$ during one iteration of the cycle.

$$\widehat{\boldsymbol{v}}_j \Leftarrow \widehat{\boldsymbol{v}}_j + \gamma(r_{ij}\widehat{\boldsymbol{u}}_i - \lambda\widehat{\boldsymbol{v}}_j) \qquad \widehat{\boldsymbol{u}}_i \Leftarrow \widehat{\boldsymbol{u}}_i + \gamma(r_{ij}\widehat{\boldsymbol{v}}_j - \lambda\widehat{\boldsymbol{u}}_i)$$

In the logical process of creating such an algorithm, both offsets are calculated before reassigning the vectors, as to not corrupt the data. This approach has a fairly simple implementation, and relatively fast run-time, making it the most popular approach towards approximating the minimization problem.

An alternative approach towards minimizing our system is Alternating Least Squares (ALS). By fixing one vector, $\widehat{\boldsymbol{u}}_i$ or $\widehat{\boldsymbol{v}}_j$, we can solve the system optimally as a Least Squares problem [9]. We can then use our solution as the fixed value in calculating the alternative vector. Each iteration of this process will decrease our error until convergence. Although ALS has a more complex implementation and run-time than SGD, implicit data is captured more accurately in the ALS optimization. This is due in part to vectors $\widehat{\boldsymbol{u}}_i$ and $\widehat{\boldsymbol{v}}_j$ being computed independently of both each other and other user or item vectors. If we were to include implicit data within our model, like mouse-activity or confidence levels, using ALS would ensure these inputs would be independent of the those for other user or item vectors, minimizing the noise within our model.

Both of these methods will minimize our system, leading to an approximation of what our complete ratings matrix may look like. Another beneficial aspect of these techniques inspired by matrix decomposition is that it is quite simple to add other inputs to capture different aspects of our data set. In the next section we will discuss adding biases and other inputs to our model.

Additional Parameters

So we have used correlations within our known ratings to predict what movies other people may like. However, we can do much better with the data given to us by Netflix. In the model we have now, all movies and users are weighted the same. Imagine we have a two different users; one is quite critical and typically rates movies below their average rating, while the other is more generous and gives movies a rating above their average. We can

introduce biases into our model to account for such behavior when making a recommendations.

For example lets define b_i as the bias of user i. That is, how much on average user i's rating of a movie differ from the movie's average rating. Lets define the bias of movie j as b_j, or how much on average a person's rating of this movie differs from the person's average rating. We can now define our predicted rating, $\widehat{k}_{ij}$, from user i of item j as

$$\widehat{k}_{ij} = b_i + b_j + \langle \widehat{\boldsymbol{v}}_j, \widehat{\boldsymbol{u}}_i \rangle$$

Since both biases b_i and b_j are dependent on vectors $\widehat{\boldsymbol{u}}_i$ and $\widehat{\boldsymbol{v}}_j$ we will have to account for the error in our minimization problem. Our system now becomes,

$$\min_{\widehat{\boldsymbol{v}},\widehat{\boldsymbol{u}}} \sum_{k_{ij} \in K} (k_{ij} - \widehat{k}_{ij})^2 + \lambda(\|\widehat{\boldsymbol{v}}_j\|^2 + \|\widehat{\boldsymbol{u}}_j\|^2 + {b_i}^2 + {b_j}^2)$$

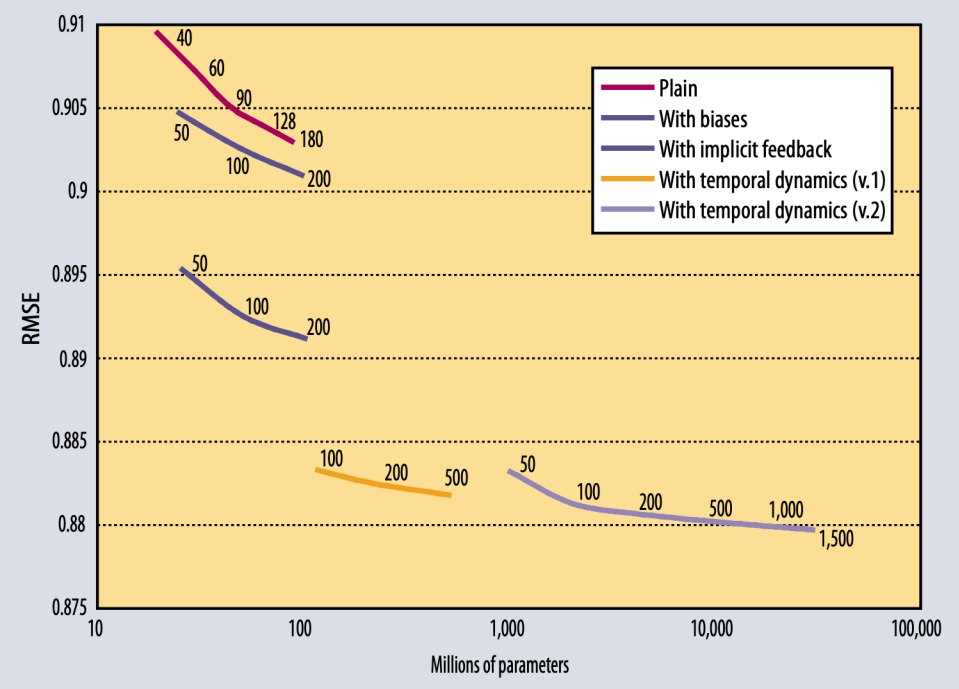
There are a variety of ways to define these biases. In some cases, using a weighted average may be more accurate. If, for example, our data set has several users that only rated a few movies, it may not capture that users actual rating bias. The Netflix data set contains movies with as little three ratings. So few ratings may not capture the item's actual average rating. Weighting the averages within our biases by how many movies a user has rated, or how many ratings a movie has, may more accurately capture trends within the data. Some models also include the global average of known ratings as a biasing parameter.

Another additional input we could use to improve our model is varying confidence levels [3]. This is just a factor of how confident we are in a certain approximation. If for some reason we believe the data for a certain item or user is not accurate, we can account for that when determining our error.

The Netflix data set contains additional information we have not used yet. With each known review is the date the review was given. So far our model has been static, but we could make our inputs dependent on time, also known as temporal dynamics [2]. A user's movie preference may change over time, so we can make our prediction dependent on when we are making the prediction to more accurately captures a users current preferences. With this information we can create a considerably accurate prediction. In fact, in 2010 Netflix cancelled a second competition due to privacy concerns, as two researchers from The University of Texas at Austin were able to identify

individual users by matching the data sets with film ratings on the Internet Movie Database.

The following is a figure created by Y. Koren, R. Bell, and C. Volinsky [2], which displays the root-mean-squared error of the systems we have discussed and others, dependent on the number of parameters used.



Clearly the more accurate models are much more complex, as they require substantially more parameters. However we were able to greatly increase the accuracy of our predictions.

Conclusion

We have seen how the SVD can capture correlations within a data set, and how it can be used for data compression. There are a number of different applications for these topics. We saw how concepts of matrix decomposition inspired data scientists in the Netflix Prize Competition, and some of the techniques used to take on the challenge.

References

[1] R. Salakutdinov, A. Mnih, G. Hinton, "Restricted Boltzmann Machines for Collaborative Filtering," University of Toronto, Canada, 2007.

[2] Y. Koren, R. Bell, C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," IEE Computer Society, 2009

[3] Y.F. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," Proc. IEEE Int'l Conf. Data Mining (ICDM 08), IEEE CS Press, 2008

[4] S. Funk, "Netflix Update: Try This at Home," 2006;
http://sifter.org/ simon/journal/20061211.html

[5] P. Monogioudis , "The Netflix Prize and Singular Value Decomposition," New Jersey Institute of technology, 2021;
https://pantelis.github.io/cs301/docs/common/lectures/recommenders/netflix/

[6] M.W. Mahoney, P. Drineas, "CUR matrix decompositions for improved data analysis," Proceedings of the National Academy of Sciences, 2009

[7] Brunton, Kutz, Proctor, "Singular Value Decomposition and Principal Components Analysis," University of Washington, 2015

[8] S. Gower, "Netflix Prize and SVD," University of Puget Sound, 2014

[9] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," Warsaw University, Poland, 2007