Recommender System with SVD

Marco Galano, s338658

Abstract
In this report, we discuss the recommender system obtained with the SVD. The aim of this system is to recommend an item (e.g. a movie, a book, etc.) to a user, based on similar items that other users have interacted with. SVD is particularly useful in this job, as we can use it to identify the meaningful number of dimensions that we would need to perform this analysis. Also, by reducing the dimensionality keeping only the dimensions related to the highest singular values, we also remove some of the noise that could create problems in our analysis.

Contents

1 Introduction **1**

2 SVD **1**
2.1 Bidiagonalization 2
2.2 Algorithm for SVD 2
2.3 Singular values analysis 2
2.4 Computing film similarities . . . 3

3 Results **3**
3.1 Toy dataframe 3
3.1.1 Dataframe analysis . . . 3
3.2 Random generated dataframe . 3

4 Additional considerations **5**

1 Introduction

For the analysis of the usage of SVD for the recommender system, we use two dataframes. The first one that we use is a toy dataframe, to understand how our analysis is behaving compared to the straightforward ideas we could get by looking at the values. The other is a randomly generated dataframe, consisting of m *users* and n *items*. The values are 1 if the user has interacted with the item, 0 instead. For simplicity we will now refer as movies to the items, so the values will be 1 if a user has watched a movie, 0 instead. The random dataframe has been generated doing the following: For each user choose a subset of watched movies, up to $\frac{n}{2}$, and assign 1 to them. The rows of the dataframe represent users, while the columns are the movies. By construction, $m \geq n$. The similarities between movies are represented by users that have watched those, so for example, we could say that if $Movie_i$ is very similar to $Movie_j$, our algorithm should recommend $Movie_j$ to a new user that has just watched $Movie_i$. Doing so with a huge catalogue of movies, and a lot of users in a platform would be computationally expensive, and the representation of the movies will be affected by noise, so we use SVD to keep just k dimensions, but still keeping the characteristics of our matrix, and enhancing the performances of our algorithm. The approximation error done is equal to σ_{k+1}, as demonstrated by the theorem of *Low Rank Approximation*. We will use $m \geq n$ to build our dataframe.

2 SVD

2.1 Bidiagonalization

To make less computationally expensive to calculate the singular values of a matrix $\mathbf{A}$, this matrix is bidiagonalized, using the Householder algorithm. Later, to compute the SVD of A we will use $\mathbf{B}$, the bidiagonalized matrix, as we know that $\mathbf{A}^T\mathbf{A}$ is similar to $\mathbf{B}^T\mathbf{B}$, so their eigenvalues (i.e. the singular values of $\mathbf{A}$) are the same.

2.2 Algorithm for SVD

The following operations are computed to perform the SVD:

1. obtain $\mathbf{B}$ and $\mathbf{H}$, respectively the bidiagonalized matrix and the Householder matrix from the matrix $\mathbf{A}$;
2. compute the eigenvalues and eigenvectors of $\mathbf{B}^T\mathbf{B}$. $\widetilde{\mathbf{Q}}$ will be the orthogonal matrix of eigenvectors, sorted with respect to decreasing respective eigenvalues;
3. $\mathbf{Q} = \mathbf{H}\widetilde{\mathbf{Q}}$;
4. $\mathbf{C} = \mathbf{A}\mathbf{Q}$;
5. compute the QR factorization of $\mathbf{C} = \mathbf{U}\mathbf{R}$;

At the end of this algorithm the matrices of the SVD decomposition of $\mathbf{A}$ will be $\mathbf{U} = \mathbf{U}$, $\Sigma = \mathbf{R}$ and $\mathbf{V}^T = \mathbf{Q}^T$. We do not need any permutation of the columns of $\mathbf{U}$ and $\mathbf{R}$ as we created $\mathbf{Q}$ by sorting the eigenvectors w.r.t. the relative decreasing order of the respective eigenvalues. However, we computed the QR factorization using numpy's library, so we need to check the sign of $\mathbf{R}$ to make sure its values are all positive.

2.3 Singular values analysis

To decide what would be the optimal value for k, that is the number of dimensions that we will use after the reduction, we plot the singular values. For example for our toy dataset we obtain, as shown in figure 1 (ignoring the drop from the first to the second), that after the third value we have a big drop. So we can say that keeping only 3 dimensions is suited for this dataframe. We can now approximate the rank of the original matrix to k = 3, so we keep the first k columns of the matrix $\mathbf{U}$, so its rows will represent the users, and the first k rows of the matrix $\mathbf{V}^T$, so its columns will represent the movies.

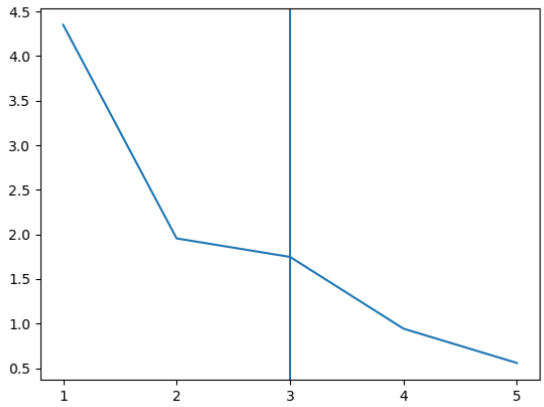

Figure 1: Singular values of toy dataframe

2.4 Computing film similarities

The function used to compute similarities between movies takes as an input a given movie, represented by a column of $\mathbf{V}^T$. We check the similarity between this column and all the other columns of the matrix. The metric used is the norm of the columns, so we will order the result according to the smallest norms obtained, and we will give as output the desired number of recommendations.

---

3 Results

Here we present the results obtained with the discussed model.

3.1 Toy dataframe

3.1.1 Dataframe analysis

The dataframe consists of 10 users and 5 movies, and it is represented by the following matrix:

$$\begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{matrix}$$

We can see that columns 3 and 4 are very similar, so a user that has just watched $Movie_3$ might also enjoy $Movie_4$. We use the algorithm with k = 3 as already said, and, we get the following recommendations list for a person that has watched $Movie_3$:

$$\mathbf{r} = \begin{bmatrix} 4 & 5 & 1 & 2 \end{bmatrix}$$

As expected, the most recommended movie is 4, so this respects our prevision. Also, computing the distance between the column of $Movie_3$ with all the other movies in the original dataframe following the order of recommendations (so we are not losing any information now), we get the following distance vector:

$$\mathbf{d} = \begin{bmatrix} 1 & 2 & 2.24 & 2.45 \end{bmatrix}$$

Showing that our model has perfectly represented the similarities of the movies in the matrix.

3.2 Random generated dataframe

The random generated dataframe (using numpy.random.seed(0) for reproducibility) has the following plot for singular values:

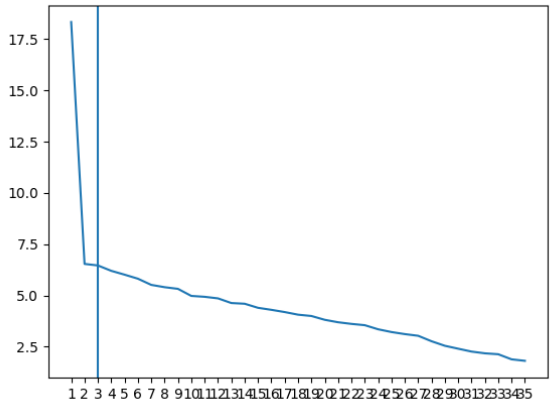

Figure 2: Singular values of the random dataframe.

We can see that the line is pretty steep everywhere. However, we can try to identify a bigger elbow from the third singular value of the plot. We will use k = 3. The recommendation list containing 5 items we get for $Movie_0$ in this case is:

$$\mathbf{r} = \begin{bmatrix} 19 & 28 & 1 & 4 & 24 \end{bmatrix}$$

The actual distances (the one obtained from the original dataframe) of this sequence of movies from $Movie_0$ are:

$$\mathbf{d} = \begin{bmatrix} 5.48 & 6 & 5.83 & 5.57 & 5.66 \end{bmatrix}$$

To get the goodness of this result, we compare this distances to those of the last 5 movies recommended, that are:

$$\mathbf{notr} = \begin{bmatrix} 27 & 20 & 12 & 33 & 29 \end{bmatrix}$$

The distance of $Movie_0$ from these movies is:

$$\mathbf{notd} = \begin{bmatrix} 6.33 & 6.40 & 6.86 & 6.78 & 6.71 \end{bmatrix}$$

As we can see, these are all larger than the ones of the recommended vector, so we could say that our model did a pretty good job. Also, to have a measure of the actual "distances" of these movies, we decided to perform a PCA with 3 PCs on the transposed matrix, that is, the matrix having as rows the movies and as columns the users. The result is the following:

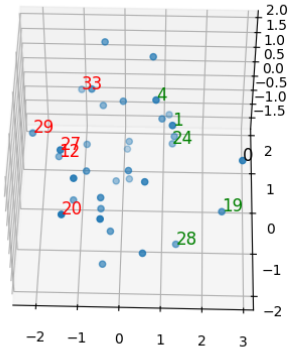

Figure 3: PCA with 3 PCs performed on the original dataframe.

We plotted the recommended movies in green, and the not recommended ones in red. We could say that also the PCA justifies our result, that is pretty meaningful, as SVD is indeed used to perform the dimensionality reduction in PCA.

---

4 Additional considerations

We have put ourselves in the situation of recommending a movie to a user that has just watched a similar one. However, meaningful analysis can also be done on the users of the platform. Let us now think that the aim of the platform is to make users keep in touch with people with similar interests: from the toy dataframe we can see that rows 1 and 8 are equal, so these users have watched the same movies. If we had to recommend a user to $User_1$, we could try to make the previous analysis, but this time we would use the matrix $\mathbf{U}$ of the SVD. On the toy dataframe this analysis produces the following: 3 recommended users for $User_1$ is:

$$\mathbf{r} = \begin{bmatrix} 8 & 6 & 10 \end{bmatrix}$$

And the actual distances in the original dataframe are:

$$\mathbf{r} = \begin{bmatrix} 0 & 2 & 1.73 \end{bmatrix}$$

As expected, $User_8$ is the most recommended one for $User_1$.