NLA project: SVD applications

Arnau Escapa

January 22, 2018

In this project of numerical linear algebra we are going to work on two different applications of the singular value descomposition(SVD) of a matrix.

1 Image compression

SVD can be used as a tool to compress images in order to save memory. We can also generate lower memory cost images at expenses of losing resolution. In the following exercise we proof the result that links SVD with image compression.

Exercise 1

The best k-rank approximation of a matrix. Let $A = USV^T$ be the SVD of a m-by-n matrix A, where $m > n$. (There are analogous results for $m < n$.) Denote by $u_1, \dots, u_n$ and $v_1, \dots, v_n$ the columns of U and V repectively. Then the matrix A_k of rank $k < n$ such that $||A - A_k||$ is minimum respect the 2-norm and Frobenius norm is given by $A_k = \sum_i^k \sigma_i u_i v_i^T$.

Observe that in this notation

$$A = USV^T = U \cdot \sum_i^n \text{diag}(0, \dots, 0, \sigma_i, 0, \dots, 0) \cdot V^T =$$

$$= \sum_i^n U \cdot \text{diag}(0, \dots, 0, \sigma_i, 0, \dots, 0) \cdot V^T = \sum_i^n \sigma_i u_i v_i^T$$

Analagously $A_k = US_kV^T$ with $S_k = \text{diag}(\sigma_1, \dots \sigma_k, 0, \dots, 0)$.

Recall the following properties that we have seen in class:

i) Rank A is equal to the number of its non-zero singular values.

ii) Given a matrix S and orthogonal matrices Q_1, Q_2 such that the product Q_1SQ_2 is well defined. Then $||Q_1SQ_2|| = ||S||$ for the 2-norm and the Frobenius norm.

iii) $||\text{diag}(\sigma_1, \dots \sigma_k)||_2 = \max_i(\sigma_i)$

iv) $||A||_F = (\sum_i^n \sigma_i)^{1/2}$

Proof. Rank$(A_k) = k$ by the fact than $A_k = US_kV^T$, and property (i).

We also have that

$$A - A_k = \sum_{i=1}^{n} \sigma_i u_i v_i^T - \sum_{i=1}^{k} \sigma_i u_i v_i^T = \sum_{i=k+1}^{n} \sigma_i u_i v_i^T = U\text{diag}(0, \ldots, 0, \sigma_{k+1}, \ldots, \sigma_n)V^T$$

For the 2-norm using property (ii) and (iii) we have

$$||A - A_k||_2 = \sigma_{k+1}$$

Let us show that there is no k-rank matrix B such that $||A-B||_2 < \sigma_{k+1}$. Let be B any m-by-n k-rank matrix, then $\dim(\text{kern}(B)) = n - k$. Let h be a unit vector of the intersection between kern(B) and the space spanned by $\{v_1, \ldots, v_{k+1}\}$ which are the first $k+1$ singular vectors of A. The insersection is not empty since the sum of their dimensions is $(n-k)+(k+1) > n$. Then, we have

$$\begin{aligned} ||A - B||_2 &= ||A - B|| \cdot ||h||_2 \geq ||(A - B)h||_2 = ||Ah||_2 = \\ &= ||USV^T h||_2 = ||S(V^T h)||_2 = (\sum_{i=0}^{n} \sigma_i^2 (v_i h_i)^2)^{1/2} = \\ &= (\sum_{i=0}^{k+1} \sigma_i^2 (v_i h_i)^2)^{1/2} \geq \sigma_{k+1} (\sum_{i=0}^{k+1} (v_i h_i)^2)^{1/2} = \sigma_{k+1} \end{aligned}$$

where in the first line we used general norm properties and the fact that $Bh = 0$ and $||h|| = 1$. In the second line we used the property (ii) and applied the 2 vector norm. And finally we applied the definition of h. We have seen $||A - B||_2 < \sigma_{k+1}$ as we wanted to see.

Let us now work on the Frobenius norm. Observe first that the rows of A_k are the projections of the rows of A to the subspace V_k spanned by the first k singular vectors of A because if a is an arbitrary row vector, since the v_i are orthogonal, the projection of the vector a to V_k is given by $\sum_{i=1}^{k} (av_i)v_i^T$.Then, the matrix whose rows are the projections of the rows of A to V_k is given by $\sum_{i=1}^{k} Av_i v_i^T$. This last expression is

$$\sum^{k} Av_i v_i^T = \sum_{i}^{k} \sigma_i u_i v_i^T = A_k$$

Now let B be the k-rank matrix that minimizes $||A-B||_F^2$. Let V be the space spanned by the rows of B. The dimension of V is at most k. Since B minimizes $||A - B||_F^2$, it must be that each row of B is the projection of the corresponding row of A to V , otherwise replacing the row of B with the projection of the corresponding row of A into V does not change V and hence the rank of B but would reduce $||A-B||_F^2$. Since each row of B is the

projection of the corresponding row of A, it follows that $||A - B||_F^2$ is the sum of squared distances of rows of A to V . Since A_k minimizes the sum of squared distance of rows of A to any k-dimensional subspace, it follows that $||A - A_k||_F^2 \leq ||A - B||_F^2$.

□

Exercise 2

In this exercise we used the previous result to generate lossy compressed images from .jpeg graphic files. We computed the SVD descomposition of the matrix given by the pixels of an image. Then applied exercise 1 to compute a lower approximation rank and generated a new .jpg image with the reduced matrix.

The memory required for storing the A_k k-rank approximation matrix is lower than storing the original matrix of pixels $A \in \mathbb{R}^{m \times n}$. Observe that the memory cost of the original matrix is $m \cdot n$ and it only takes $(m + n) \cdot k$ to store $u_1, \ldots, u_k$ and $\sigma_1 v_1, \cdots, \sigma_k v_k$. (which is enough to reconstruct A_k). Hence for low values of k, storing A_k is cheaper.

The file `img-compression.py` contains two routines that implement this strategy for image compression using SVD. The prototype of the first routine is

```
image_compresion_BW(file_path,K)
```

where `file_path` is the path of the .jpg image we want to compress and `K` is a list with the rank values of the approximate matrices A_k. If the image is a black and white image, the routines computes the A_k approximation to create a .jpg compressed file for each value k. They are stored in the same path that the original file and their names are the same as the original but they also have the percentage of the Frobenius norm captured in each compressed file. In the case, the original image is a color RGB image the routine also generates `len(K)` black and white image compressions. To do so, it computes de A_k approximation of the first component of the RGB coordenates.

The second routine has the following prototype

```
image_compresion_color(file_path,K)
```

and it is analogous to the first routine but it implements the compression for color images by performing the A_k approximations for each of the RGB components of the image. To compute the percentage of the Frobenius norm captured in each compressed file we computed the mean of the Frobenius norm captured in each RGB component approximation.

The program `img-compression.py` use this routines to perform several experimental images compressions. The program ouputs via terminal the

size of the images and the required time to perform the computations. The original and generated images can be found on the `IMG` directory.

2 Principal component analysis

Principal component analysis is a technique to detect the main components of a data set in order to reduce it into fewer dimensions retaining the relevant information.

In the following exercices we implemented PCA for two different data sets by taking advantage of the SVD descomposition of the data. The PCA consists essensially in finding an orthogonal base so that each dimension preserves the maximum variance of the data. For each direction one can quantify the ratio of total variance of a data set $X \in \mathbb{R}^{n \times m}$. This value can be found by the SVD of

$$Y = \frac{1}{\sqrt{n-1}} X^T$$

because the portion of the total vairance in each of the principal components is given by $s_i^2 / \sum_i s_i^2$ where s_i are the singular values of Y.

Another important information is the coordinates of the data by the PCA basis. It can be shown that they are given by $V^T X$ where V^T is the reduced SVD descomposition of $Y = USV^T$.

When applying PCA one assumes that the data is measured in comparable physical units. For this reason, we will center our data by subtracting the mean of the observations for each variable. In this way we will be working on the covariance matrix. When assuming that data is Gaussian distribuited we will also starterize data for each variable and then we will be working with the correlation matrix.

Once the principal components are found and the respective portions of the total variance are computed there are several criterias to decide how many of this components are enough to descrive the original data set. Each particular case may require different treatment depending on its application but there exist different general criteria rules:

1. **Kaiser rule** consists in droping all components with eigenvalues under 1 (i.e singular values under 1).

2. **Scree plot**. Consist in ploting the components as the X axis and the corresponding eigenvalues as the Y-axis. As one moves to the right, toward later components, the eigenvalues drop. When the drop ceases and the curve makes an elbow toward less steep decline, Cattell's scree test says to drop all further components after the one starting the elbow. This rule is sometimes criticised for being amenable to

researcher-controlled "fudging". That is, as picking the "elbow" can be subjective because the curve has multiple elbows or is a smooth curve, the researcher may be tempted to set the cut-off at the number of factors desired by their research agenda. [Wikipedia]

3. **Variance criteria**: Some researchers simply use the rule of keeping enough factors to account for a certain amount of the variation. Here we will use a 3/4 of the total variance.

Data Set 1

The file `PCA-ex1.py` implements the PCA analysis for the `example.dat` dataset using both the covariance and the correlation matrices. Since our data is dense and we want to compute all singluar values we used `numpy.linalg.svd` in order to compute the SVD descomposition of Y. `PCA-ex1.py` outputs via terminal the portion of the total variance accumulated in each of the principal components, the standard deviation of each of the principal components and the expression of the original dataset in the new PCA coordinates. Since the amount of data is small we tested the result by computing the sorted eigenvalues of the covariance matrix $C_x = \frac{1}{n-1}XX^T$. We obtained the following result:

```
Using covariance matrix:

   Variance for component:
 [ 0.7613434   0.19408757  0.04456903]
   SD for component:
 [ 6.72580753  3.39588526  1.62731162]
   Dataset in the new PCA coordinates:
 [[-8.41413327 -2.41101716  5.91600212  4.90914832]
 [ 2.80841544 -4.93431485  1.28847753  0.83742188]
 [-0.05161413 -0.16441793 -1.87836222  2.09439429]]

Result tested

Using correlation matrix:

   Variance for component:
 [ 0.77349665  0.17461965  0.05188369]
   SD for component:
 [ 4.06217453  1.93008271  1.0520704 ]
   Dataset in the new PCA coordinates:
 [[-5.47956461 -0.6171141   3.55459126  2.54208745]
 [ 1.18180892 -2.85004614  0.47030906  1.19792817]
 [-0.24780803  0.22672546 -1.25585536  1.27693792]]
```

Let us discuss how many components are necessary to represent the data. Using Kaiser rule we count how many singluar values of S are greater than one. In this case, 3.

The Scree plot consists in plotting the eigenvalues of C_x , which are given by the square of the singluar values of Y. We also plotted the line $\lambda = 1$

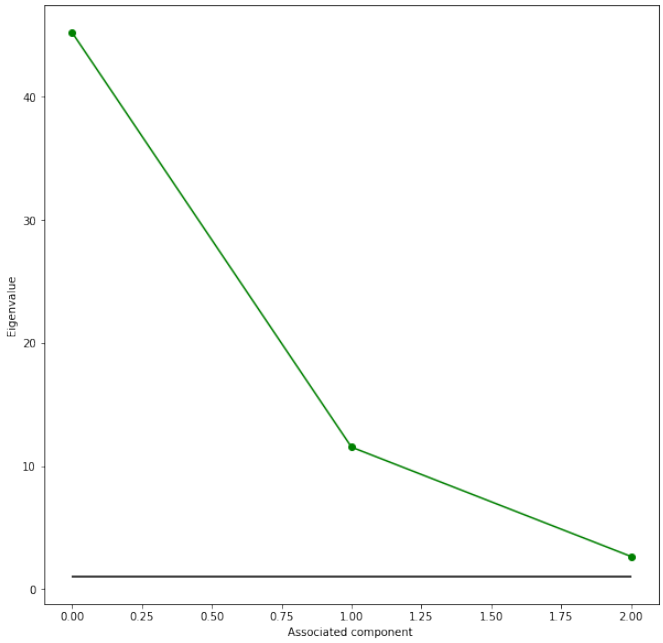

Figure 1: Scree plot of data set 1.

According to the Scree plot one would take 2 or 3 components.

And finally using the variance rule at 3/4 one would choose just one component. However given the low dimension of the original data and the fact that the two first components add to 95% of the total variance one may prefer taking two components.

Data Set 2

In this case we performed the PCA analysis for a larger data set, `RCsGoff.csv`. The file `PCA-ex2.py` implements the PCA using the covariance matrix and stores the results in `PCA-ex2-output.txt` with the following format:

Sample,PC1,PC2,...,PC20,Variance

where Sample stands for day0 rep1,...,day18 rep3 (i.e. the different observations) and PCi stands for the coordinate of the principal component of the observation. Finally variance is the portion of the total variance accumulated in each of the principal components. Given the sparse structure

of the original data we used the function `sparsesvd` of the library with the same name (see the reference). This routine performs faster than `numpy.linalg.svd` for sparce matrices and, particulary for our matix Y. In addition to that it requires less memory because it takes advantage of the CSC format for sparse matrices.

We use the output file to plot the data using the two more rellevant principals components.

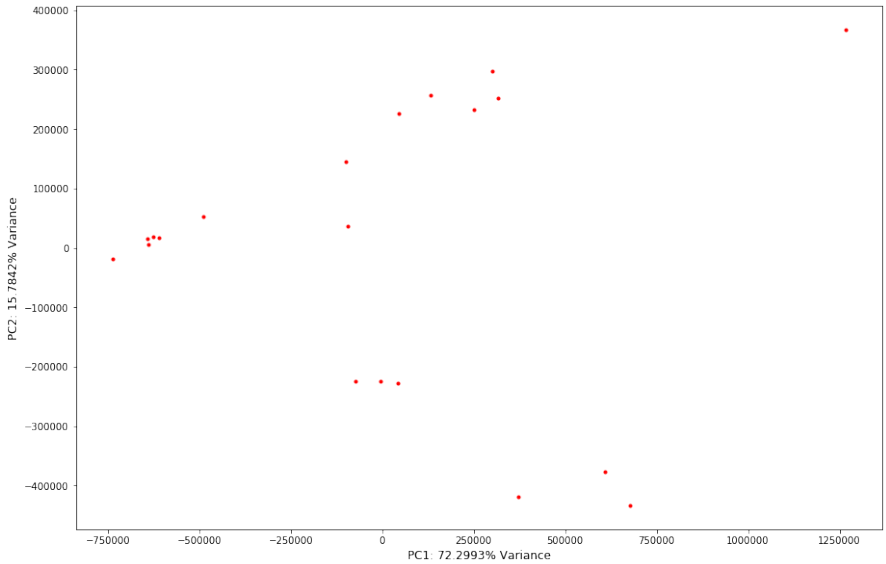

Figure 2: Plot of the data using PC1 and PC2

Let us discuss how many components are necessary to represent the data. The relevant components according to Kaisers rule are those with associated eigenvalue with argument greater than 1.

The non-nul eigenvalues are

```
2.75450879e+11   6.01358971e+10   2.54840201e+10   1.10007261e+10
4.67185920e+09   1.75752114e+09   1.38183861e+09   6.00220854e+08
1.56916033e+08   1.19517084e+08   4.47010974e+07   4.04761481e+07
3.90349263e+07   3.08606427e+07   1.97053601e+07   1.66439968e+07
1.53172413e+07   1.12461888e+07   9.75378866e+06   3.32203234e+03
```

Hence we take 18 dimensions from the original 20.

The Scree plot is

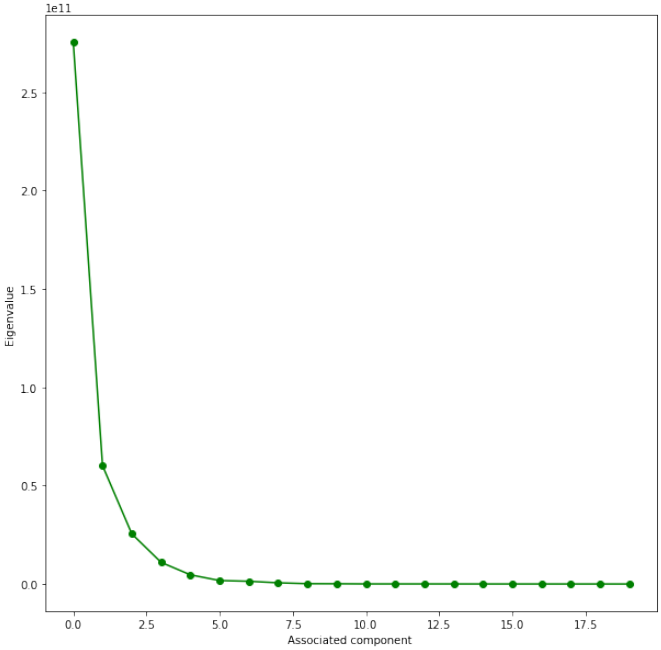

Figure 3: Eigenvalues of C_x.

According the Scree plot one would take 3 or 4 components.

To apply the variance 3/4 rule we plot the accumulated variance for component toghether with a y line at 75%.

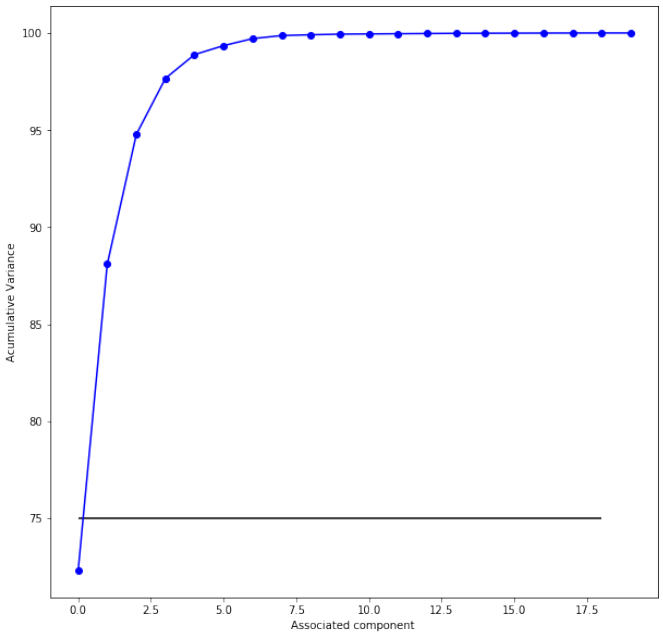

Figure 4: Accumulated variance.

Hence, according to this rule we would take 2 components.

In conclusion, PCA can be a powerful tool but the choise of the significant number of components may be problematic. The existing rules techniques are quite heuristic , not very fiable. They are useful for having an idea of how much is the variance concentrated but one must be careful when dropping data and one should not obey this rules blindly.