

Lab 2: Matrix Decompositions

Prof. Gaston Sanchez

Stat 154, Fall 2017

Introduction

In this lab you will be working with the functions `svd()` and `eigen()` to perform singular value decomposition, and eigenvalue decomposition, respectively.

In addition, you will have to write code in R to implement the Power Method, which is a basic procedure to obtain the dominant eigenvector (and related eigenvalue) of a square matrix.

The content of this lab will introduce you to a series of concepts and computations that will constantly emerge throughout the course. So it makes more sense to start learning first about these two fundamental decompositions (SVD & EVD) before tackling any of the statistical learning tasks (both supervised and unsupervised).

Data USArrests

In this lab we are going to use the data set `USArrests` that comes in R—see `?USArrests` for more information. This data set is about “Violent Crime Rates by US State”, and contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

```
head(USArrests)
```

##	Murder	Assault	UrbanPop	Rape
## Alabama	13.2	236	58	21.2
## Alaska	10.0	263	48	44.5
## Arizona	8.1	294	80	31.0
## Arkansas	8.8	190	50	19.5
## California	9.0	276	91	40.6
## Colorado	7.9	204	78	38.7

The four variables are:

- **Murder:** numeric Murder arrests (per 100,000)
- **Assault:** numeric Assault arrests (per 100,000)
- **UrbanPop:** numeric Percent urban population
- **Rape:** numeric Rape arrests (per 100,000)

Singular Value Decomposition

Let \mathbf{M} be an $n \times p$ matrix of full column-rank p (assume $n > p$). The singular value decomposition (SVD) of \mathbf{M} is given by:

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

where:

- \mathbf{U} is an $n \times p$ matrix of left singular vectors
- \mathbf{D} is a $p \times p$ diagonal matrix of singular values
- \mathbf{V} is a $p \times p$ matrix of right singular vectors

Function `svd()`

R provides the function `svd()` that allows you to compute the SVD of any rectangular matrix. Here's a basic example with `USArrests`:

```
SVD <- svd(USArrests)
```

The default output of `svd()` is a list with three elements:

- `u`: matrix of left singular vectors
- `v`: matrix of right singular vectors
- `d`: vector of singular values

You can use the arguments `nu` and `nv` to have more control over the behavior of `svd()`:

- `nu`: the number of left singular vectors to be computed.
- `nv`: the number of right singular vectors to be computed.

SVD on raw data

- Use `svd()` to compute the SVD of `USArrests`
- Take the output of `svd()` and create the matrices \mathbf{U} , \mathbf{D} , and \mathbf{V}
- Confirm that the data of `USArrests` can be obtained as the product of: $\mathbf{U}\mathbf{D}\mathbf{V}^T$

SVD and best Rank-one Approximations

As we saw in lecture, one of the most attractive uses of the SVD is that it allows you to decompose a matrix \mathbf{M} as a sum of rank-one matrices of the form: $l_k \mathbf{u}_k \mathbf{v}_k^T$. This is the result of the famous [Eckart-Young-Mirsky](#) theorem, which says that the best rank r approximation to \mathbf{X} is given by:

$$\mathbf{X}_r = \sum_{k=1}^r l_k \mathbf{u}_k \mathbf{v}_k^T$$

where l_k is the k -th singular value, that is, the k -th diagonal element in \mathbf{D} .

Consequently, the best $r = 2$ rank approximation to \mathbf{X} is:

$$\mathbf{X}_2 = l_1 \mathbf{u}_1 \mathbf{v}_1^T + l_2 \mathbf{u}_2 \mathbf{v}_2^T$$

In other words, \mathbf{u}_1 , and \mathbf{u}_2 form the best 2-dimensional approximation of the objects in \mathbf{X} (living a p -dim space).

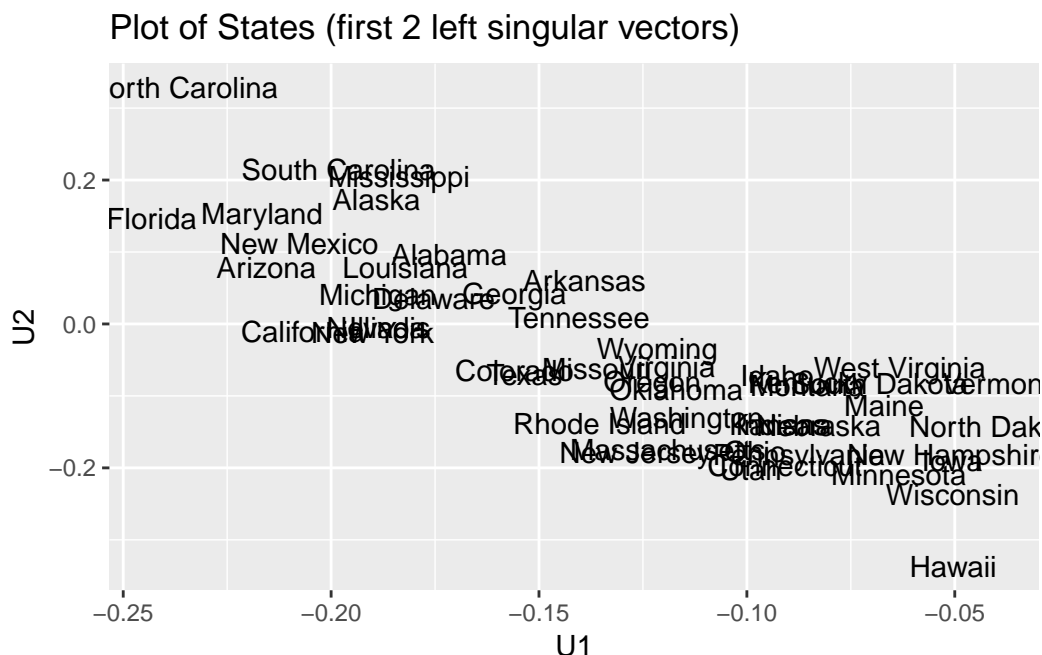
Your turn

- Confirm that the sum: $\sum_{k=1}^4 l_k \mathbf{u}_k \mathbf{v}_k^T$ equals `USArrests`
- Create a new variable (vector) `MA` by adding `Murder + Assault`
- Create a new data.frame `Arrests2` with the variables `Murder`, `Assault`, `UrbanPop`, `Rape`, and `MA`.
- Compute the SVD of `Arrests2`
- How does the singular values of `USArrests` compare to those of `Arrests2`?
- What is the rank of `Arrests2`?

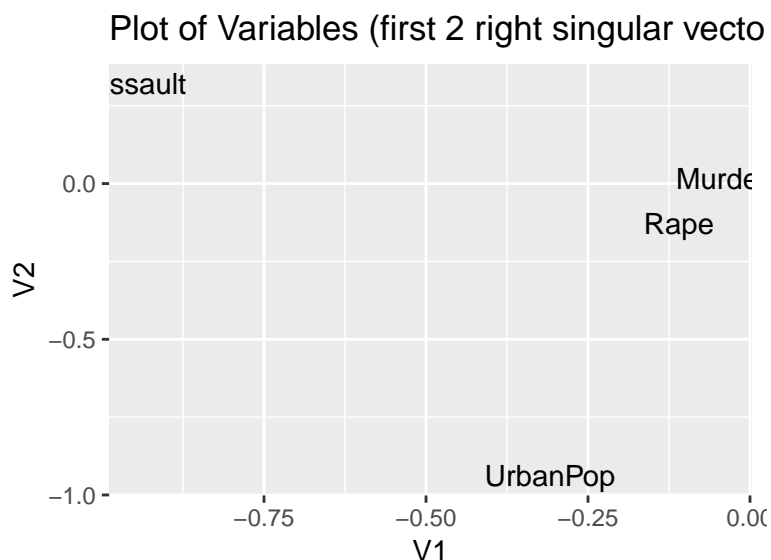
Lab continues on next page

Using SVD output to visualize data

Knowing that we can approximate `USArrests` with $l_1 \mathbf{u}_1 \mathbf{v}_1^T + l_2 \mathbf{u}_2 \mathbf{v}_2^T$, use \mathbf{u}_1 and \mathbf{u}_2 to visualize the States with a simple scatterplot. Create your own scatterplot like the one below:



In the same way, use the first two right singular vectors of \mathbf{V} to graph a scatterplot (e.g. 2-dimensional representation) of the variables:



Later in the course, we will see how to get—and interpret—better visualizations (based on some kind of SVD or EVD factorization).

Eigenvalue Decomposition

R provides the function `eigen()` to compute the spectral decomposition—better known as *eigenvalue decomposition*—of a square matrix.

For multivariate analysis and statistical learning techniques, the typical square matrices are some variation of the sum-of-squares and cross-products: $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$. For instance, assuming that variables in \mathbf{X} are standardized (mean = 0, var = 1), the (sample) correlation matrix \mathbf{R} is based on the product $\mathbf{X}^T\mathbf{X}$, that is: $\mathbf{R} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$.

```
R <- cor(USArrests)
```

The output of `eigen()` is a list of two elements, `values` and `vectors`.

```
# EVD of correlation matrix
evd <- eigen(R)
```

You can be more specific and use the argument `symmetric = TRUE` to indicate that the input matrix is symmetric (and don't let R guess it by itself):

```
evd <- eigen(R, symmetric = TRUE)
```

If you need only the eigenvalues you can use the parameter `only.values = TRUE`:

```
eigenvalues <- eigen(R, symmetric = TRUE, only.values = TRUE)
eigenvalues
```

```
## $values
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
##
## $vectors
## NULL
```

Inverse of covariance matrix

- Without using `scale()`, compute a matrix \mathbf{X} as the mean-centered data of `USArrests`.
 - Calculate the sum-of-squares and cross-products matrix $\mathbf{S} = \mathbf{X}^T\mathbf{X}$
 - \mathbf{S} is proportional to the (sample) covariance matrix $\frac{1}{n-1}\mathbf{X}^T\mathbf{X}$. Confirm that `cov(X)` is equal to $\frac{1}{n-1}\mathbf{X}^T\mathbf{X}$.
 - Use `solve()` to compute the inverse \mathbf{S}^{-1}
 - Use `eigen()` to compute the EVD of $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$
 - Confirm that \mathbf{S}^{-1} can also be obtained as: $\mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^T$
-

Power Method

The **Power Method** is an iterative procedure for approximating eigenvalues.

First assume that the matrix \mathbf{A} has a dominant eigenvalue with corresponding dominant eigenvector. Then choose an initial approximation \mathbf{u}_0 (must be a non-zero vector) of one of the dominant eigenvectors. This choice is arbitrary (and in theory should work with almost any vector). Then, form the sequence $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k$, given by:

$$\begin{aligned}\mathbf{u}_1 &= \mathbf{A}\mathbf{u}_0 \\ \mathbf{u}_2 &= \mathbf{A}\mathbf{u}_1 = \mathbf{A}^2\mathbf{u}_0 \\ \mathbf{u}_3 &= \mathbf{A}\mathbf{u}_2 = \mathbf{A}^3\mathbf{u}_0 \\ &\vdots \\ \mathbf{u}_k &= \mathbf{A}\mathbf{u}_{k-1} = \mathbf{A}^{k-1}\mathbf{u}_0\end{aligned}$$

For large powers of k , and by **properly scaling** this sequence, you will see that you obtain a good approximation \mathbf{u}_k of the dominant eigenvector of \mathbf{A} .

Here's an example. Consider the following matrix

$$\mathbf{A} = \begin{pmatrix} 2 & -12 \\ 1 & -5 \end{pmatrix}$$

Let's start with an initial nonzero vector \mathbf{u}_0

$$\mathbf{u}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

In R, we create \mathbf{A} and \mathbf{u}_0

```
# matrix
A <- matrix(c(2, 1, -12, -5), nrow = 2, ncol = 2)
A

##      [,1] [,2]
## [1,]    2  -12
## [2,]    1  -5

# starting vector
u0 <- c(1, 1)
```

Let's see what happens in the first four iterations:

```

u_old <- u0
for (k in 1:4) {
  u_new <- A %*% u_old
  print(paste('iteration =', k))
  print(u_new)
  # replace v_old
  u_old <- u_new
}

```

```

## [1] "iteration = 1"
##      [,1]
## [1,]  -10
## [2,]   -4
## [1] "iteration = 2"
##      [,1]
## [1,]   28
## [2,]   10
## [1] "iteration = 3"
##      [,1]
## [1,]  -64
## [2,]  -22
## [1] "iteration = 4"
##      [,1]
## [1,]  136
## [2,]   46

```

Note that the obtained vectors seem to be very different from one another. However, the difference is in the scale. This is why the part “properly scaling” is very important. How do you scale \mathbf{u}_k ? Well, there are different options, but probably the simplest one is by using the L_∞ -norm.

L_p -norms reminder:

- L_1 -norm: $\sum_{i=1}^n |u_i|$
- L_2 -norm: $\sqrt{\sum_{i=1}^n (u_i)^2}$
- L_∞ -norm: $\max\{|u_1|, \dots, |u_n|\}$
- L_p -norm: $(\sum_{i=1}^n |u_i|^p)^{1/p}$

Description of Power Method

Here is the procedure of the Power Method to find the largest eigenvalue and its corresponding eigenvector. Write code in R to implement such method.

1. Start with an arbitrary vector \mathbf{u}_0

2. Iteration for a series of steps $k = 0, 1, 2, \dots, n$ to form the series of \mathbf{u}_k vectors:

$$\mathbf{u}_{k+1} = \frac{\mathbf{A}\mathbf{u}_k}{s_{k+1}}$$

where s_{k+1} is the entry of $\mathbf{A}\mathbf{u}_k$ which has the largest absolute value (this is actually a scaling operation dividing by the L_∞ -norm).

3. When the scaling factors s_k are not changing much, s_{k+1} will be close to the eigenvalue and \mathbf{u}_{k+1} will be close to an eigenvector.
4. Check your answer to see if it worked well. If things didn't work well, try another \mathbf{u}_0 . You were miraculously unlucky in your choice.

Find the largest eigenvalue of the following matrix

$$\mathbf{A} = \begin{pmatrix} 5 & -14 & 11 \\ -4 & 4 & -4 \\ 3 & 6 & -3 \end{pmatrix}$$

Compare your results with those provided by `eigen()`.

Other scaling options

The scaling step in the power method:

$$\mathbf{u}_{k+1} = \frac{\mathbf{A}\mathbf{u}_k}{s_{k+1}}$$

involves choosing a value for s_{k+1} . One option for s_{k+1} is the entry of $\mathbf{A}\mathbf{u}_k$ with the largest absolute value (i.e. L_∞ -norm). But you can actually use other scaling strategies. For instance, you can use the L_1 -norm or the L_2 -norm.

Modify your code to use any other L_p -norm in the power algorithm, and confirm that you get the same dominant eigenvectors and eigenvalues.

Deflation and more eigenvectors

In order to get more eigenvectors and eigenvalues, you need to apply the Power Method on the residual matrix obtained by **deflating** \mathbf{A} with respect to the first eigenvector. This deflation operation is:

$$\mathbf{A}_1 = \mathbf{A} - \mathbf{u}_1 \mathbf{u}_1^\top$$

where \mathbf{u}_1 is the first eigenvector.