Lab 2: Matrix Decompositions

Stat 154, Spring 2018

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 `state.x77`

In this lab we are going to use the data set `state.x77` that comes in R—see `?state.x77` for more information. `state.x77` is part of a collection of data sets about "US State Facts and Figures".

```
head(state.x77)
```

```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20
## Alaska            365   6315        1.5    69.31   11.3    66.7   152
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65
## California      21198   5114        1.1    71.71   10.3    62.6    20
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166
##              Area
## Alabama     50708
## Alaska     566432
## Arizona    113417
## Arkansas    51945
## California 156361
## Colorado   103766
```

More precisely, `state.x77` is a matrix with 50 rows and 8 columns giving the following statistics in the respective columns.

- `Population`: population estimate as of July 1, 1975
- `Income`: per capita income (1974)
- `Illiteracy`: illiteracy (1970, percent of population)
- `Life Exp`: life expectancy in years (1969–71)
- `Murder`: murder and non-negligent manslaughter rate per 100,000 population (1976)
- `HS Grad`: percent high-school graduates (1970)
- `Frost`: mean number of days with minimum temperature below freezing (1931–1960) in capital or large city
- `Area`: land area in square miles

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:

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

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 `state.x77`

- Take the output of `svd()` and create the matrices **U**, **D**, and **V**
- Confirm that the data of `state.x77` can be obtained as the product of: $\mathbf{UDV}^\mathsf{T}$

```
## [1] TRUE
```

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: $l_k \mathbf{u_k} \mathbf{v_k}^\mathsf{T}$. 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:

$$\mathbf{X_r} = \sum_{k=1}^{r} l_k \mathbf{u_k} \mathbf{v_k}^\mathsf{T}$$

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

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

$$\mathbf{X_2} = l_1 \mathbf{u_1} \mathbf{v_1}^\mathsf{T} + l_2 \mathbf{u_2} \mathbf{v_2}^\mathsf{T}$$

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).

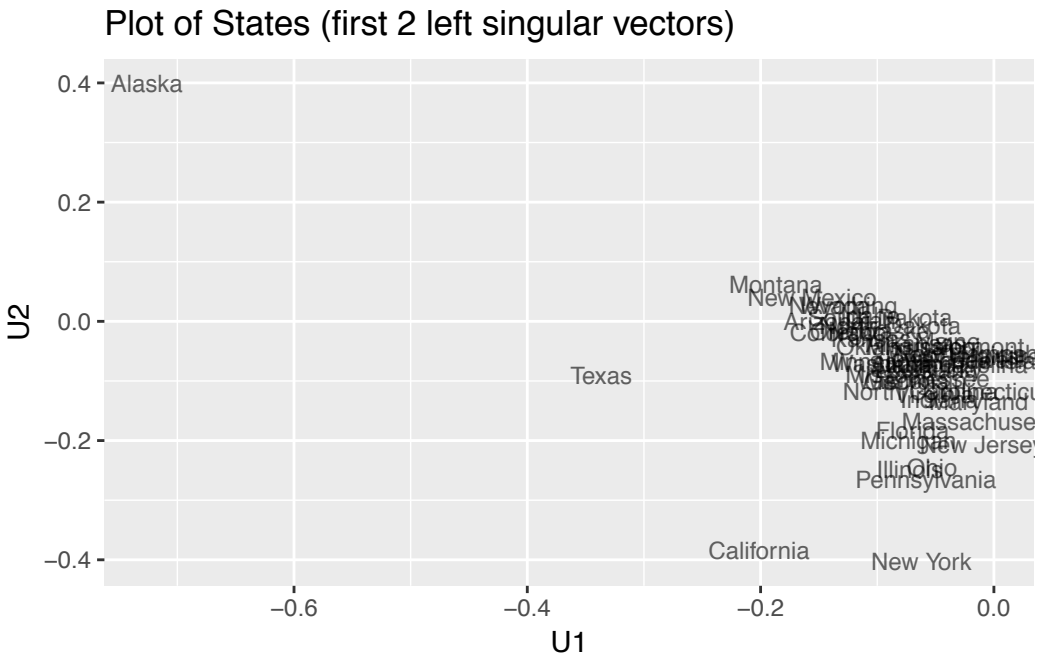
Your turn

- Create an object `state2` by selecting (i.e. subsetting) the first five columns of `state.x77`
- Perform the SVD decomposition of `state2`
- Confirm that the sum: $\sum_{k=1}^{5} l_k \mathbf{u_k} \mathbf{v_k}^\mathsf{T}$ equals `state2`

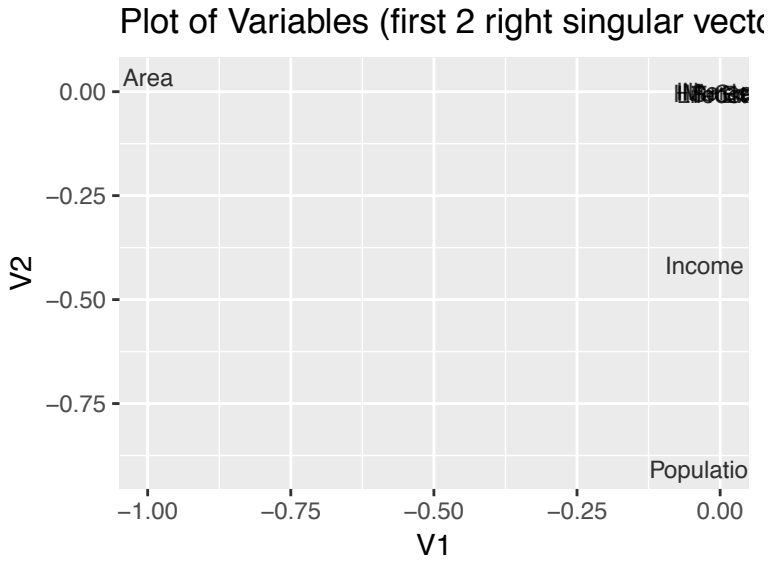
Lab continues on next page

Using SVD output to visualize data

Consider the SVD decomposition of `state.x77`. Knowing that we can approximate `state.x77` 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: $\mathbf{X}^\intercal\mathbf{X}$ and $\mathbf{X}\mathbf{X}^\intercal$. 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 $\mathbf{X}^\intercal\mathbf{X}$, that is: $\mathbf{R} = \frac{1}{n-1}\mathbf{X}^\intercal\mathbf{X}$,.

```
R <- cor(state.x77)
```

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

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

```
## [1] "values"  "vectors"
```

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] 3.5988956 1.6319192 1.1119412 0.7075042 0.3846417 0.3074617 0.1444488
## [8] 0.1131877
##
## $vectors
## NULL
```

Inverse of covariance matrix

- Without using `scale()`, compute a matrix $\mathbf{X}$ as the mean-centered data of `state.x77`.
- Calculate the sum-of-squares and cross-products matrix $\mathbf{S} = \mathbf{X}^\intercal\mathbf{X}$
- $\mathbf{S}$ is proportional to the (sample) covariance matrix $\frac{1}{n-1}\mathbf{X}^\intercal\mathbf{X}$. Confirm that `cov(X)` is equal to $\frac{1}{n-1}\mathbf{X}^\intercal\mathbf{X}$.
- Use `solve()` to compute the inverse $\mathbf{S}^{-1}$
- Use `eigen()` to compute the EVD of $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\intercal$
- Confirm that $\mathbf{S}^{-1}$ can also be obtained as: $\mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^\intercal$

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 $\mathbf{w}_0$ (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 $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k$, given by:

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

For large powers of k, and by **properly scaling** this sequence, you will see that you obtain a good approximation $\mathbf{w}_k$ 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 $\mathbf{w}_0$

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

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

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

```
# starting vector
w0 <- c(1, 1)
```

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

```
w_old <- w0

for (k in 1:4) {
  w_new <- A %*% w_old
  print(paste('iteration =', k))
```

```
  print(w_new)
  cat("\n")
  # update w_old
  w_old <- w_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 $\mathbf{w}_k$? Well, there are different options, but probably the simplest one is by using the L_∞-norm.

Reminder of L_p-norms:

- L_1-norm: $\sum_{i=1}^{n} |w_i|$
- L_2-norm: $\sqrt{\sum_{i=1}^{n} (w_i)^2}$
- L_∞-norm: $max\{|w_1|, \ldots, |w_n|\}$
- L_p-norm: $\left(\sum_{i=1}^{n} |w_i|^p\right)^{1/p}$

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 $\mathbf{w}_0$

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

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

   where s_{k+1} is the entry of $\mathbf{A}\mathbf{w}_k$ 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 largest eigenvalue of $\mathbf{A}$, and $\mathbf{w}_{k+1}$ will be close to the eigenvector associated to s_{k+1}.

4. You can also veirfy that s_{k+1} will be very close to the eigenvalue given by the Rayleigh quotient:

$$\lambda \approx \frac{\mathbf{w}_{k+1}^\mathsf{T}\mathbf{A}\mathbf{w}_{k+1}}{\mathbf{w}_{k+1}^\mathsf{T}\mathbf{w}_{k+1}}$$

Your turn: Implement the power method to 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()`. Keep in mind that the eigenvectors of `eigen()` have unit Euclidean norm (i.e. L_2-norm). Likewise, recall that the eigenvectors are only defined up to a constant: even when the length is specified they are still only defined up to a scalar.

Other scaling options

The scaling step in the power method:

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

involves choosing a value for s_{k+1}. One option for s_{k+1} is the entry of $\mathbf{A}\mathbf{w}_k$ 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 A** with respect to the first eigenvector. This deflation operation is:

$$\mathbf{A}_1 = \mathbf{A} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^\mathsf{T}$$

where λ_1 is the first eigenvalue and $\mathbf{v}_1$ is the corresponding eigenvalue.

Deflate the matrix $\mathbf{A}$ and apply the power method on $\mathbf{A}_1$ to obtain more eigenvalues and eigenvectors.