First Approach to Principal Component Analysis in DNA Array Design

Myrna I. Merced Serrano
Department of Mathematics
University of Puerto Rico at Humacao
100 Tejas Avenue
Humacao, Puerto Rico 00791
mymese@mate.uprh.edu

Faculty Adviser: José O. Sotero Esteva

Keywords: DNA Arrays, Principal Components Analysis

Abstract

In this work the method of Principal Components Analysis (PCA) is used to reduce dimensions of datasets formed by genomic data. A sophisticated version of this technique may be used later as a help for the design of DNA arrays. DNA arrays have applications to many areas of biology and medicine, such as studying treatments, disease, and developmental stages. To test the program the first input that we used was some books downloaded from the Internet. Then we tested the program with some sequences of genomes obtained from NCBI's Gene Bank. A graphical representation of the dimesion-reduced data is inspected to assess the effectiveness of the method. After comparing the performance of the PCA algorithm on the text books and the Salmonella complete genomes, we obtain the reasonably good results. Data from similar sources is visually allocated in the same area.

1. Introduction

Principal Components Analysis (PCA) is a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. The objective of principal component analysis is to reduce the dimensionality (numbers of variables) of the data set but retain the most of the original variability in the data. The first principal component accounts for as much the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible[1].

The PCA performed on the symmetric covariance matrix or on the symmetric correlation matrix. These matrix be calculated from the data matrix.

The mathematical technique used in PCA is called eigen analysis we solve for the eigenvalues and eigenvectors of the squares symmetric matrix with sum of squares and crossproducts. The eigenvector associated with the largest eigenvalue determines the direction of the second principal component. The sum of the eigenvalues equals the trace of the square matrix and the maximum number of eigenvalues equals the number of rows (or columns) of this matrix[2].

A DNA microarray is a collection DNA segments (probes) attached to a solid surface, such as silicon chip, forming an array for the purpose of monitoring of expression levels for thousands of genes simultaneously. DNA microarrays have applications to many areas of biology and medicine, such as studying treatments, disease, and developmental stages.

2. Background

2.1 covariance[3]

The covariance is an important part of this project. It is based on the measure of the linear dependence of $\mathbf{X_1}$ and $\mathbf{X_2}$ where $\mathbf{X_1}$ and $\mathbf{X_2}$ are column vectors that have the frequency of the symbols. This quantity, defined over the bivariate population associated with $\mathbf{X_1}$ and $\mathbf{X_2}$, is called the covariance of $\mathbf{X_1}$ and $\mathbf{X_2}$.

In the notation the covariance is

$$Cov(X_1, X_2) = E[(X_1 - u_1)(X_2 - u_2)] = \frac{1}{N-1} \sum_{i=1}^{n} (X_{1j} - u_{1j})(X_{2j} - u_{2j}),$$

where $u_1 = E(X_1) = \frac{1}{N-1} \sum_{i=1}^{n} X_{1j}$ and $u_2 = E(X_2) = \frac{1}{N-1} \sum_{i=1}^{n} X_{2j}$.

The larger the absolute value of the covariance of $\mathbf{X_1}$ and $\mathbf{X_2}$, the greater the linear dependence between $\mathbf{X_1}$ and $\mathbf{X_2}$. Positive values indicate that $\mathbf{X_1}$ increases as $\mathbf{X_2}$ increases; negative values indicate that $\mathbf{X_1}$ decreases as $\mathbf{X_2}$ increases. A zero value of the covariance would indicate no linear dependence between $\mathbf{X_1}$ and $\mathbf{X_2}$.

2.2 eigenvalues and eigenvectors[4]

The study of linear operators on infinite-dimensional spaces is an important part of mathematics known as functional analysis. We concerned with finding eigenvalues and eigenvectors of linear operators that are defined on finite-dimensional vector spaces. The following theorem serves as a first step in this direction.

Theorem

Suppose $\boldsymbol{A}$ is an n x n matrix and $\boldsymbol{T}$: $R^n \to R^n$ is defined by $\boldsymbol{T}(\mathbf{v}) = \boldsymbol{A}\mathbf{v}$. Then the real number λ is an eigenvalue of $\boldsymbol{T}$ if and only if **det** (λ **I** – $\boldsymbol{A}$) = 0.

For a linear map $\boldsymbol{T}$: $R^n \to R^n$ is defined in terms of multiplication by an n x n matrix $\boldsymbol{A}$, this theorem says that the eigenvalue of $\boldsymbol{T}$ are precisely the real solutions of the equation **det** (λ **I** – $\boldsymbol{A}$) = 0. As you will soon discover, the formula **det** (λ **I** – $\boldsymbol{A}$) expands to a sum of various constant multiples of 1, λ, λ^2, λ^3, …, λ^n. Therefore, the formula **det** (λ **I** – $\boldsymbol{A}$) defines a polynomial of degree n in the variable λ . Here is some relevant terminology.

Suppose ***A*** is an n x n matrix. The n^{th} degree polynomial in the variable λ defined by

$$\mathbf{det}\,(\lambda\,\mathbf{I} - \boldsymbol{A})$$

is the characteristic of ***A***. The real zeros of the characteristic equation $\mathbf{det}\,(\lambda\,\mathbf{I} - \boldsymbol{A})$ are called eigenvalues of matrix ***A***. A nonzero vector $v \in R^n$ such that $(\lambda\,\mathbf{I} - \boldsymbol{A})\mathbf{v} = 0$ is an eigenvector of ***A*** associated with λ. The solution space

$$E_A(\lambda) = v \in R^n \mid (\lambda I - A)\, v = 0$$

of this homogeneous system is the eigenspace of ***A*** associated with λ.

Be sure to notice that the eigenvalues of a linear operator are required to be real numbers even though the characteristics equation may have solutions that are complex numbers with nonzero imaginary parts.

3. Methods and software

In this work the method of Principal Components Analysis (PCA) is used to reduce dimensions of datasets. This method is based on solving some matrix equations. To store and manipulate these matrices we use array constructions provided by the NumPy (Numeric Python) extension to the Python language.

3.1 making arrays

NumPy consists of a set of extensions to the Python programming language which allow Python programmers to efficiently manipulate large sets of objects organized in grid-like fashion[5]. These sets of objects are called arrays and they can have any number of dimensions. The version 24.2 of NumPy it was that we use[6].

To be able to use NumPy the following instructions were used:

```
import Numeric
from Numeric import *
```

NumPy provides two fundamental objects: an N dimensional array object (`ndarray`) and a universal function object (`ufunc`). To be able to use the PCA we use many of these functions since with them different types of operations with matrix can be done. Two examples of operations that we use are the multiplication of matrices (`matrixmultiply(x,y)`) and the calculation of the eigenvectors (`eigenvectors(x)`).

3.2 PCA algorithm

The PCA algorithm consists of solving a series of matrix equations.

3.2.1 *data matrix*

An initial dataset consisting of the quantities of each symbol of each input file. There is computed based on the symbols found in those file. This information is stored in a matrix **X**.

Figure 1 illustrates the algorithm to make the data matrix.

```
X = zeros((M,N), Float)
      contm = 0
      for letra in letrasAInformar:
            for n in range(Archivos):
                  if (letrasEnArchivos[n]).has_key(letra):
                        X[contm,n] = letrasEnArchivos[n][letra]
            contm += 1
```

Figure 1. Code (in NumPy) used for PCA algorithm to make the data matrix.

We declare **N** as the quantity of columns, one to each file, and **M** as the quantity of the rows of **X**, one for each symbol. We do not want entries of any row to be all equal. Those rows do not provide usefull information to distinguish between files and may produce divisions by zero as we will see later. Therefore we use a function that finds any such row and another that erases it. Figure 2 illustrates the `delete_row()` function that is used to erase the equal rows and Figure 3 show the `eliminaFilasIguales()` function that is used to eliminate the rows that have the same frequency in the matrix **X**.

```
def delete_row(matrix, row):
   return Numeric.take(matrix, range(row) + range(row+1, matrix.shape[0]))
```

Figure 2. Code (in NumPy) it is used for erase a row of a matrix.

```
def eliminaFilasIguales(X):

      M=X.shape[0]
      for i in range(M-1,0,-1):
            f = take (X, (i,))
            if (max(f[0]) == min(f[0])):
                  X = delete_row(X, i)
                  M = M - 1
      return X
```

Figure 3. Code (in NumPy) it is use to eliminate a row of a matrix.

3.2.2 *normalization of data*

Then we normalize the matrix **X**. Normalization ensures that the differences among the length of the files not be considered. For example, we have a file that contains 100 letters, divided into 50 A, 30 B and 20 C; and another file that contains the double of each of those letters. When we normalize we can see that each symbol in the files are relatively equally important. Figure 4 shows the function that is utilized to normalize.

```
def normalizar(X):
      [M,N]=X.shape
      sx = sum(X)
      for i in range(N):
            for j in range(M):
                  if (sx[i] != 0):
                        X[j][i] /= sx[i]
      return X
```

Figure 4. Code (in NumPy) it is use to normalize the matrix **X**.

3.2.3. *vector of means*

The next step in the PCA is to make the column vector of the means called **u**, one mean for each row of the matrix **X**. We use another function that can be seen in the Figure 5.

```
def promediosPorFilas(X):
      [M,N]=X.shape
      u=[]
      for i in range(M):
            acum = 0.
            for j in range(N):
                  acum += X[i,j]
            u.append (acum/N)
      u = transpose(reshape(u,(1,M)))
      return u
```

Figure 5. Code (in NumPy) it is use to compute the mean of each row.

3.2.4. *deviations from the mean*

In the next step we solve the equations that are used in PCA to reduce dimensions of the dataset. These functions compute the deviations from the mean , the covariance, the standard deviations, the projected matrix , the eigenvectors , the basis vectors and the projecting of the data onto the basis vectors.

In details, a matrix **B** is constructed to store the deviations from the mean. The equation is:

$$B = X - u * \vec{1}$$

where $\vec{1}$ is the row vector of **M** ones.

3.2.5. *covariance matrix*

The covariance matrix is now computed in this way:

$$C = \frac{1}{N-1} B B^T$$

where B^T is the transpose of **B**. The diagonal of **C** has the variance of each symbol trough the files.

3.2.6. *standard deviations vector*

We construct the column vector of standard deviations **s** by extracting square roots of this diagonal. In the figure 6 you can see the NumPy code to this instruction.

```
s = transpose(array([sqrt(diagonal(C))]))
```

Figure 6. Code (in NumPy) it is use to make the vector column **s**.

3.2.7. *eigenvalues and eigenvectors*

To be able to obtain the final product we need to seek the eigenvectors that satisfy to that assembly of data and as soon as to find them we have to order them of decreasing form. As discussed in section 2.2 the eigenvectors will provide a new set of reference vectors and the eigenvalues will determine the important of the eigenvectors. Thus, we want to select the eigenvectors corresponding to the **L** largest eigenvalues. To be able to do these operations we use functions already defined in NumPy inside of the Lineal Algebra Module. These instructions are in Figure 7.

```
evals, evects = eigenvectors(C)

pos = argsort(abs(evals),axis=-1)

evals = take(evals,pos)

V = take(evects,pos,axis=-1)
V = V.real
```

Figure 7. Code (in NumPy) it is use to obtain the eigenvectors and sort them.

Let **W** and be the matrix consisting of the first **L** vectors of **V** that is, the eigenvectors corresponding to the **L** largest eigenvalues of **C**. The dimensions of this matrix **M** x **L**. We often use **L**= 2 and **L**= 3. In this case the columns of **W** determine a plane in in the original **N** dimensional space. All the points in the dataset will be projected onto this plane.

3.2.8. *projection from N dimensions to L dimensions*

The matrix that is used to make the projection is represented by **Z**. This matrix is computed using the mean and the standard deviation of each row **M** of the matrix **X**. We need the matrix **B**, the vector **s** and the vector that is of all 1's. So the equation is:

$$Z = \frac{B}{s * \vec{1}}$$

The division is meant to be entry. Sin no entry of **s** is zero, the divisions are well defined.

The last matrix that is to utilized to do the different graphics with the dataset, is call **Y** and consists of **N** columns vectors, where each vector is the projection of the corresponding data vector from the matrix **X** onto the basis vectors contained in the columns of matrix **W**. With the functions that exist in NumPy we can do this in two steps as is shown in the Figure 8.

```
W = take(V, range(L), 1)

Y = matrixmultiply(transpose(W),Z)
```

Figure 8. Code (in NumPy) it is use to obtain the final matrix.

3.3 graphical representation of results

The columns of **Y** represents points in the **L** dimensional space. We use **L** = 2 and **L** = 3. *Gnuplot* and *Jmol* are used to draw the graphics.

3.4 to run the program

To run the program you must have kept all the files that you go to use in a same folder. Then in our case that we are using linux, we open a terminal and seek that folder. After you can see all the documents that are stored in that folder you must write this instruction in the same terminal:

```
python ProgramName
```

where `ProgramName` is the name of the program as you kept it. Then enter the input data.

3.5 input data

To test the program the first input that we use was some books thats we download from the Internet. This books are from different languages. We use 5 books of the German language, 4 books of the Spanish language and 4 books of the English language. We make graphics with the results and compared.

Then we test the program with some sequences of genomes. In specific the genomes of Salmonella enterica, Escherichia coli and Shigella. We obtain the complete sequences of this genomes from the GenBank of NCBI[7].

4. Results and discussion

In this section the results obtained by comparing the input files are presented. They were analyzed and they describe using graphics and by the different classes of input that were the books and the sequences of genomes.

4.1. analysis of books

The first results that were obtained they were based applying the PCA to the deferent books data. Figure 9 shows the graph for the PCA to the books.

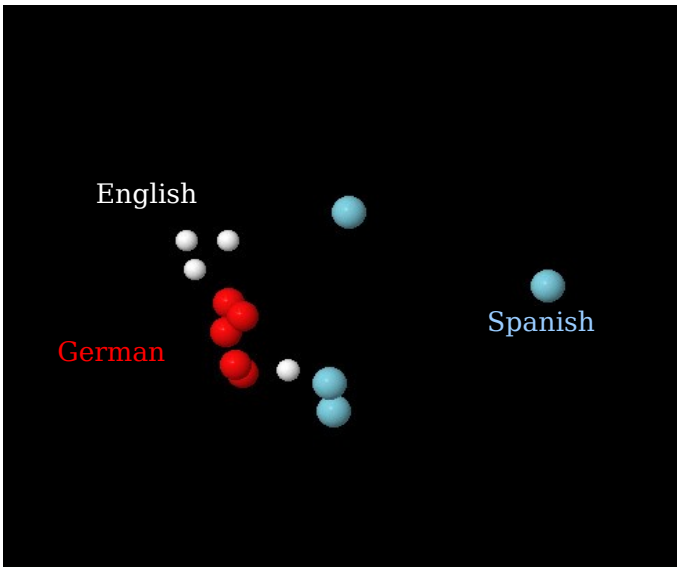

Figure 9. Graph of all books that we tested the program.

The most points of Spanish and German are agglomerated each one with its other book of the same language. The English points not are agglomerated like Spanish and Germany. It may be because we used one book of English that is for statistics and this book have much numbers than letters and this affected the frequency of the symbols for the English language.

If we see some nearness with different language points we run the program with only this languages books. So we tested also only with Germany and Spanish books and the Figure 10 shows the graph.

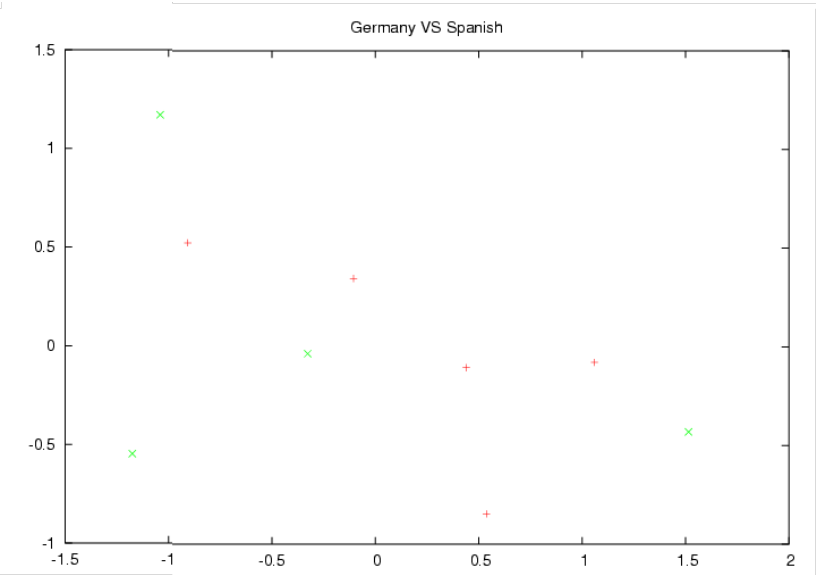

Figure 10. Graph of Germany and Spanish books.

The Spanish language is represents by (x) and the Germany by (+).

The points of this languages are switch together and we can say that Germany and Spanish have similar frequencies in the use of their symbols. Because in above graphics the points stay together.

4.2. analysis of genomes

Then we entered complete genomes as input data and applied the PCA algorithm with **L** = 3. The Figure 11 show the results for all the varieties of Salmonella that we chose.

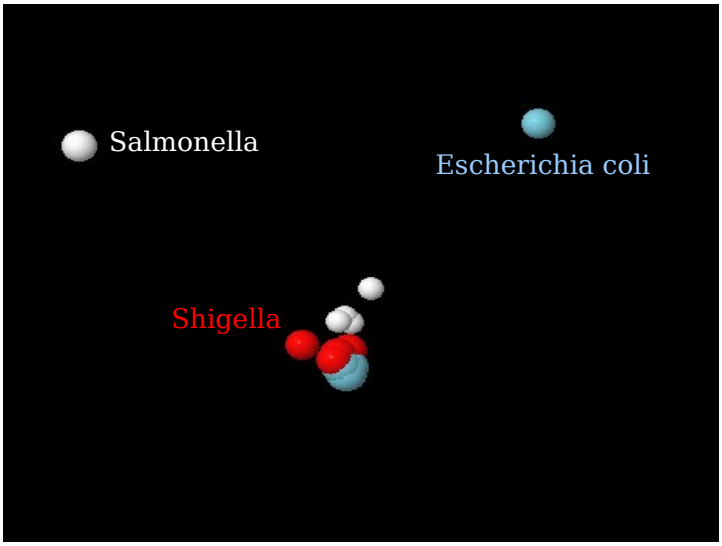

Figure 11. Graph of the varieties that we chose from Salmonella.

We can see that the white balls represents the Salmonella genomes, the red represents the Shigella, and the blue represents the Escherichia coli. The large ball that has red and blue is a representation that the points are very near when we see them in 3D. Also there they can be other points inside and therefore is that is seen so large.

5. Conclusions

After comparing the performance of the PCA algorithm on the text books and the Salmonella complete genomes, we obtain the reasonably good results. Data from similar sources is visually allocated in the same area.

6. Future Works

The future work is to beginning the design of the DNA array and then apply this array to biology research project.

7. Reference

1. Principal Components Analysis, http://www.resample.com/xlminer/help/PCA/pca_intro.htm
2. Principal Components Analysis, http://www.fon.hum.uva.nl/praat/manual/Principal_component_analysis.html
3. W. Mendenhall, Mathematical Statics with Applications
4. R. Messer, Linear Algebra Gateway to Mathematics
5. D. Ascher, “An Open Source Project Numeric Python”, http://numeric.scipy.org/.
6. Numerical Python Home Page, http://www.numpy.org/.
7. NCBI GenBank, http://www.ncbi.nlm.nih.gov/Genbank/
8. M. Lutz, Programming Python
9. D. Beazley, Python Essential Reference Second Edition
10. T. Altom, Programming with Python
11. T. Williams, “Gnuplot An Interactive Plotting Program”
12. Principal Components Analysis,http://en.wikipedia.org/wiki/Principal_components_analysis