MÉTODOS NUMÉRICOS PARA LA COMPUTACIÓN

Trabajo de curso: Análisis de Componentes Principales (PCA)

2020/21

---

26 de enero de 2021

Grupo 03: José María Amusquívar Poppe y Prashant Jeswani Tejwani

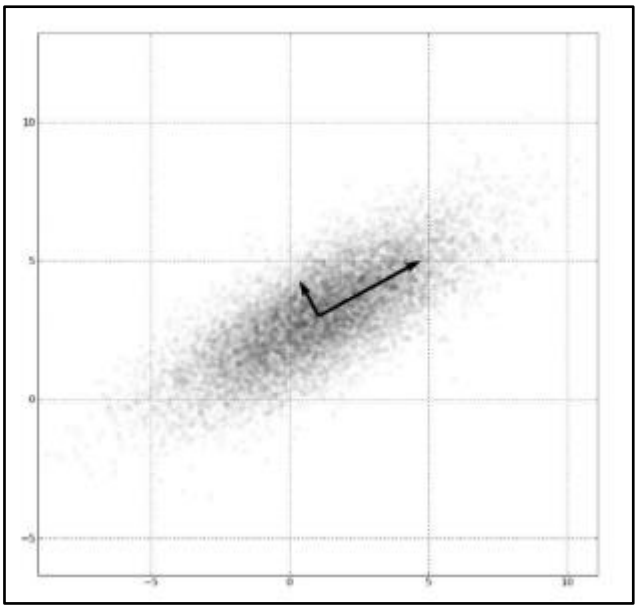
Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería en Informática

Contenido

Introducción

La técnica denominada PCA (Principal Component Analysis) es empleada en áreas como la Visión por Computador o la Minería de Datos para simplificar la representación de datos masivos y obtener una descripción más compacta de los mismos. Dado un conjunto de puntos (instancias o muestras), se desea obtener las coordenadas de esos puntos en un sistema de representación de nuevas coordenadas (dimensiones o características) que están centrada en el conjunto de datos y rotadas en un alineamiento a las direcciones principales del agrupamiento de los datos.



Estas direcciones se pueden obtener como los autovectores correspondientes a los autovalores más grandes de la matriz de correlación.

Para realizar esta práctica se ha seleccionado el conjunto de datos “Gas Turbine CO and NOx Emission Data Set Data Set”, cuyo enlace de descarga se encuentra en el apartado de Referencias. Este “dataset” está compuesto por cinco ficheros en formato CSV, donde cada uno representa diversas observaciones durante un determinado año, desde 2011 hasta el 2015, acumulando un total de 36733 observaciones, y 11 características de tipo real.

Cada una de las 11 características se corresponde con un determinado sensor que recogen muestras tales como la temperatura, humedad o presión del ambiente, las emisiones de monóxido de carbono, las emisiones de óxido de nitrógeno, entre otras.

Por tanto, puesto que, para la realización de esta práctica, únicamente, es necesario un mínimo de 200 observaciones, se ha decidido realizar el estudio del fichero de 2015, la cual contiene 7384 observaciones y 11 características.

Desarrollo

Solución con Matlab/Octave

Se ha realizado el cálculo del "PCA" usando autovectores y autovalores, como solicita el enunciado, y, además, se ha empleado la factorización "SVD" y la función integrada "PCA" para obtener el mismo cálculo, y se ha verificado que en los tres casos se obtiene el mismo resultado.

1. **Extraer del fichero de datos las características de tipo real. Se generará una matriz "X" de "m" filas (instancias) por "n" columnas (dimensiones).**

   Para poder leer el fichero de datos en formato "CSV" se ha empleado el método "readtable()". Este método permite obtener los datos en forma de "table", compuesta por sus respectivas cabeceras y los datos.

```
dataset = readtable("dataset/gt_2015.csv");
headers = dataset.Properties.VariableNames;
features = dataset.Variables;
```

Figura 1. Funciones para leer y dar formato a los datos.

   A partir de la variable "dataset", se pueden obtener las cabeceras de los datos usando el atributo "Properties.VariableNames", de igual modo, para obtener los datos como una matriz se emplea el atributo "Variables". Así pues, se procede a comprobar que las dimensiones de la matriz corresponden con los previsto (7384x11).

```
>> size(features)

ans =

        7384          11
```

Figura 2. Comprobación del tamaño del "dataset".

2. **Centrar los datos restando la media de cada componente, generando una matriz "XC".**

   Para realizar el centrado de los datos, tal como se menciona en el enunciado, se calcula la media de cada característica (columna) y, posteriormente, se itera sobre la matriz de datos por columnas, restando a cada una de ellas la media calculada de dicha columna.

```
function XC = centerData(X)
    meanFeatures = mean(X, 1);
    XC = zeros(size(X));
    for i=1:size(X, 2)
        XC(:, i) = X(:, i) - meanFeatures(i);
    end
end
```

Figura 3. Centrado de los datos usando la media de cada componente.

3. Calcular los autovalores y autovectores de la matriz de covarianza "Z = (XC'*XC)/m".

Una vez se ha obtenido la matriz de datos centrada, se procede a calcular la matriz de covarianza tal como explica el enunciado, obteniendo una matriz "Z" como resultado. Posteriormente se calculan los autovectores y autovalores de dicha matriz de covarianza empleando la función "eig()".

4. Representar los datos y los autovalores principales.

Continuando con el apartado anterior, para poder representar los datos de una manera correcta, es necesario obtener una nueva matriz cuyas columnas representen las componentes principales de los datos. Para conseguir esto, se ha de reordenar el vector de autovalores atendiendo a un orden decreciente, obtener los índices de estos cambios y reordenar, también, la matriz de autovectores a partir de los nuevos índices de los autovalores.

```
[V_, D_] = eig(Z);
[D_, order] = sort(diag(D_), 'descend');
V_ = V_(:,order);
```

Figura 4. Obtención de autovalores y autovectores, y reordenamiento.

Entonces, una vez se tiene la matriz "V_" reordenada según los autovalores, se procede a multiplicar la matriz de datos centrada por esta matriz de autovectores, obteniendo así, la nueva matriz de datos "PCA".
De los cálculos realizados arriba, se obtienen 3 parámetros, los autovalores "D_EIG", los autovectores y la matriz de datos "PCA". Estos autovalores pueden emplearse para calcular el porcentaje de información que se representará:

```
P_TOTAL = (D_EIG/sum(D_EIG)); %
percentage = sum(P_TOTAL(1:3));
```

Figura 5. Cálculo del porcentaje de información de la matriz "PCA".

En este caso, basta con representar las 3 primeras componentes principales de la matriz “PCA” para conseguir tener el 92% de los datos graficados.

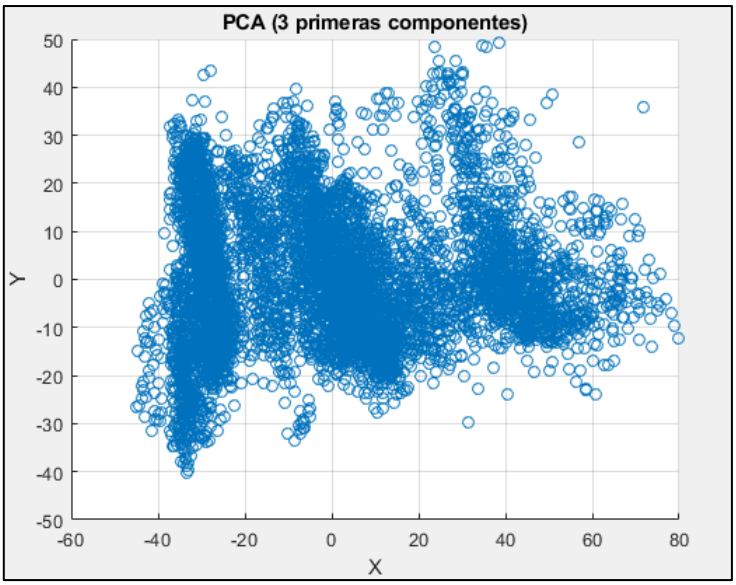


Figura 6. Gráfica "PCA" vista desde arriba.

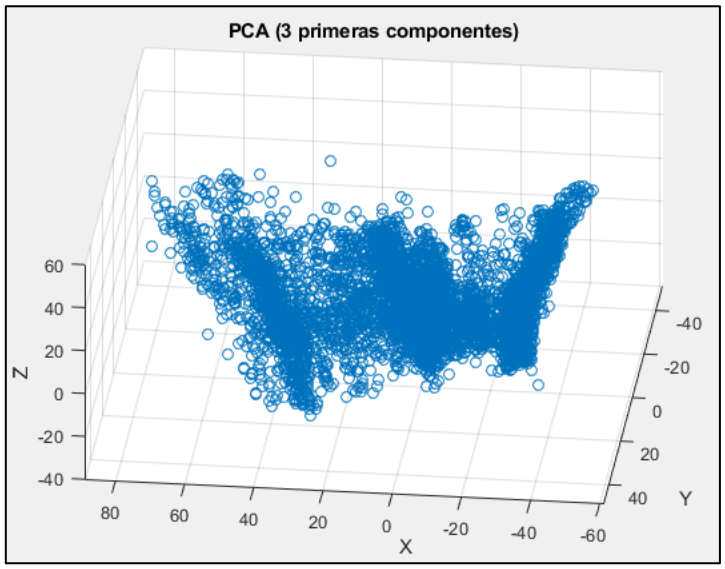


Figura 7. Gráfica "PCA" vista en tres dimensiones.

5. ¿Qué ocurre al multiplicar los datos por la matriz de autovectores?

Como los autovectores determinan la dirección del nuevo espacio de coordenadas, y los autovalores determinan su magnitud, al multiplicar la matriz de datos original y centrada por estos autovectores (reordenados según los autovalores), se obtienen las coordenadas de esos puntos en un sistema de representación de nuevas coordenadas (dimensiones o características) que están rotadas en un alineamiento a las direcciones principales del agrupamiento de los datos.

Esta matriz posee las mismas dimensiones que la matriz de datos original, sin embargo, cada columna o componente de esta nueva matriz se corresponde con una combinación lineal de las características originales abarcando una menor carga de información conforme se va desplazando de columnas hacia la derecha.
Por lo que importa emplear las primeras columnas de la izquierda para realizar cualquier representación gráfica, empleando una menor cantidad de dimensiones.

Otras soluciones [Opcional]

Se ha realizado el opcional de "OpenMP", sin embargo, dado que es acumulativo, se ha realizado antes el opcional de "Blas/Lapack", ya que es útil para realizar "OpenMP".

La estructura del proyecto está compuesta por una carpeta llamada "Datos" que contiene dentro el "dataset", y los distintos códigos fuentes. Estos códigos se organizan en 4 ficheros, uno es la raíz, "Main", desde el que se llama a los demás, otro fichero alberga el código necesario para leer el fichero de datos, "CSV_Reader", el tercero está compuesto por distintas funcionalidades comunes, "Utils", y el último alberga el código del cálculo de "PCA", tanto por autovectores y autovalores, como por factorización "SVD".

Entonces, dentro del fichero "Main" se encuentra el "main" del programa en el que, lo primero se realiza, es leer los datos pasándole la ruta del fichero y el separador, obteniendo un vector (funciona como matriz) con los datos leídos, además de obtener el número de observaciones "m" y el de características "n".

```
int main(int argc, char* argv[]){
    CSV_Reader reader(path, ',');
    double* dataRead = reader.readData();
    int m = reader.getM(); int n = reader.getN();

    double* pcaAutos = getPCAAutos(dataRead, m, n);
    double* pcaSVD = getPCASVD(dataRead, m, n);

    return 0;
}
```

Figura 8. Estructura el "main" del programa.

Dado que se ha realizado el "PCA" de los datos tanto con autovectores y autovalores como con factorización "SVD", se ha creado una función para cada uno, los cuales se encargan a llamar a cada clase ("PCA_Autos" y "PCA_SVD") para realizar el respectivo cálculo.

```
double* getPCAAutos(double * dataRead, int m, int n) {
    // Constructor que prepara los datos para el cálculo del PCA (centrado y covarianza)
    PCA_Autos pcaAutos(dataRead, m, n);
    // Se calcula autovectores y autovalores, y se reordenan de mayor a menor (Matriz de covarianza)
    pcaAutos.autos('V', 'U');
    // Se obtiene los datos mediante PCA
    return pcaAutos.getPCA();
}
```

Figura 9. Procesamiento de obtención del "PCA" con autovectores y autovalores.

La figura anterior muestra la estructura general desarrollada para el cálculo del “PCA”, ya sea usando autovectores o factorización “SVD”. Pues, lo primero que se realiza es centrar los datos y calcular la matriz de covarianza, realizado implícitamente, mediante un método privado en el constructor, al crear un objeto “PCA_Autos”. Esto genera la matriz de covarianza, que se empleará en la función “pcaAutos.autos(‘V’, ‘U’)”, con la función “LAPACKE_dsyev()”.

Y, finalmente, la función “pcaAutos.getPCA()”, que obtiene la matriz “PCA” a partir de la multiplicación de los datos por los autovectores, usando la función de “cblas_dgemm()”

- ## BLAS/LAPACK

  La función empleada para calcular los autovectores y autovalores recibe dos caracteres, “jobz” y “uplo”, en este caso, el primero ha de ser igual a ‘V’ para obtener los autovalores y autovectores, y el segundo puede ser ‘U’ o ‘L’ según donde se quiera sobrescribir. La matriz de entrada (matriz de covarianza) es “autosVectAux”, que será sobrescrita con los autovectores, mientras que el vector de autovalores se almacena en “autosValAux”.

```
res = LAPACKE_dsyev(layout, jobz, uplo, n_C, autosVectAux, lda, autosValAux);
```

Figura 10. Función que computa los autovalores y autovectores de la matriz.

La función que se emplea para realizar la factorización “SVD” recibe dos caracteres, “jobu” y “jobvt”, en este caso, ambos serán igual a ‘A’, pues se quieren guardar todas las columnas de los vectores singulares “u”, y la matriz “vt”. Los valores singulares son almacenados en “s”, y la matriz de entrada (matriz de covarianza) corresponde a la variable “matrixOver”.

```
res = LAPACKE_dgesvd(layout, jobu, jobvt, m_C, n_C, matrixOver, lda, s, u, ldu, vt, ldvt, superb);
```

Figura 11. Función que computa los valores y vectores singulares de la matriz.

La función que se encarga de multiplicar los datos de entrada centrados por los autovectores recibe datos como la trasposición de las matrices, sus dimensiones, los escalares “alpha” que será igual a 1, y el escalar “beta” que será igual a 0, pues no se requiere realizar una suma con la matriz “C” (“dataPCA”), esta matriz simplemente almacenará el resultado.

```
cblas_dgemm(layout, transA, transB, m, n, k, alpha, data, lda, autosVect, ldb, beta, dataPCA, ldc);
```

Figura 12. Función que realiza la multiplicación de matrices.

- ## OpenMP

  También se ha hecho uso de OpenMP para utilizar el paralelismo multihilo. Por ejemplo, a la hora de centrar los datos, se calcula la media de cada columna y posteriormente se resta la media correspondiente a cada componente de su columna, esto es realizado anteriormente con 3 bucles. Al implementar OpenMP, ha decidido dividir el cálculo de la siguiente manera: como el conjunto de datos está formado por 11 columnas/características, se crean 11 secciones las cuales son ejecutadas por un hilo distinto el cual se ocupará de

calcular la media de una columna y a continuación, restar la media a cada uno de sus componentes. Debido a que el ordenador del laboratorio cuenta con 8 hilos, habrá hilos que ejecute más de una sección. Se le ha añadido la cláusula “nowait” ya que una vez que termine de calcular y restar la media de una columna, ningún hilo tiene la necesidad de esperar a que acabe otro.

```
#pragma omp parallel num_threads(11)
{
    int id = omp_get_thread_num();

    #pragma omp sections nowait
    {
        #pragma omp section
        {
            //printf("-- [Proceso %d] -- [section 1]\n", id);
            double sum = 0;
            for (int i = 0; i < m; i++) {
                sum += data[i * n];
            }
            means[0] = sum / m;

            for (int i = 0; i < m; i++) {
                data[i * n] -= means[0];
            }
        }
```

Figura 13. Se paraleliza la función que centra los datos.

Finalmente, en los otros bucles se ha paralelizado con la directiva “for” el cual comparte las iteraciones del bucle entre todos los miembros de un equipo de hilos. El equipo de hilos se ha establecido a 8 hilos. No se ha determinado cómo se dividirán las iteraciones de los bucles (para dividir la matriz entre “m” para obtener la matriz de covarianza y reordenar los autovalores de mayor a menor) entre hilos con el “schedule”, sino se ha dejado que lo determina la CPU.

```
// Reordenar los autovalores de mayor a menor
#pragma omp parallel num_threads(8)
{
    int cont = 0;
    //#define CHUNK 2
    #pragma omp for nowait
    for (int i = m_C - 1; i >= 0; i--) {
        autosVal[cont] = autosValAux[i];
        cont++;
    }
}
```

Figura 14. Se paraleliza la reordenación de los autovalores.

```
#pragma omp parallel num_threads(8)
{
    // Z[11x11] Matriz de covarianza global
    //#define CHUNK 11
    #pragma omp for nowait
    for (int i = 0; i < n_ * n_; i++) {
        Z[i] /= m_;
    }
}
```

Figura 15. Se paraleliza la división del cálculo de la matriz de covarianza.

Conclusión

Para concluir este informe, se presenta a continuación una comparación de los autovectores y autovalores obtenidos tanto en “MATLAB” como en “C++”. De este modo se certifica que ambos proyectos funcionan de un modo correcto y similar.

```
Autovalores
  718.1627  190.6965  131.3372   46.8905   24.1666   14.7351    1.8690    1.2841    0.1921    0.0180    0.0022

Autovectores
   -0.1053    0.3747   -0.3963   -0.1544    0.0025   -0.7836   -0.0132    0.0644    0.2215    0.0148   -0.0023
    0.0181   -0.1426    0.2799    0.7460    0.4678   -0.3443    0.0679    0.0231    0.0448    0.0009   -0.0004
    0.1597   -0.8706   -0.3754   -0.1640    0.0524   -0.2106    0.0359    0.0124    0.0150    0.0041   -0.0002
   -0.0212   -0.0002   -0.0058    0.0011   -0.0122   -0.0378   -0.0196    0.0035   -0.0889   -0.9947   -0.0142
   -0.1529   -0.0177    0.0718   -0.0238   -0.1683   -0.0267    0.9696    0.0078    0.0332   -0.0161   -0.0002
   -0.7306   -0.0679   -0.0102   -0.2215    0.4024   -0.0260   -0.0338   -0.0594   -0.4903    0.0563   -0.0409
    0.0967    0.1213   -0.1875   -0.2387    0.6970    0.3151    0.1380   -0.1420    0.5007   -0.0706    0.0616
   -0.5806   -0.2342    0.2351    0.0380   -0.2662    0.0275   -0.1811   -0.0234    0.6667   -0.0426   -0.0082
   -0.0411   -0.0116    0.0125    0.0057   -0.0288   -0.0231   -0.0115    0.0056   -0.0463   -0.0078    0.9971
    0.0606   -0.0058    0.0687    0.0006   -0.0944   -0.1381   -0.0064   -0.9807   -0.0414    0.0051   -0.0008
    0.2317   -0.0822    0.7232   -0.5336    0.1465   -0.3172   -0.0163    0.0950    0.0187   -0.0005   -0.0004
```

Figura 16. Autovalores y autovectores obtenidos en "MATLAB".

```
Autovalores:
              718.1627        190.6965        131.3372        46.8905 24.1666 14.7351 1.8690  1.2841  0.1921  0.0180 0.0022

Autovectores:
              -0.1053 0.3747  -0.3963 -0.1544 0.0025  -0.7836 -0.0132 0.0644  0.2215  0.0148  -0.0023
              0.0181  -0.1426 0.2799  0.7460  0.4678  -0.3443 0.0679  0.0231  0.0448  0.0009  -0.0004
              0.1597  -0.8706 -0.3754 -0.1640 0.0524  -0.2106 0.0359  0.0124  0.0150  0.0041  -0.0002
              -0.0212 -0.0002 -0.0058 0.0011  -0.0122 -0.0378 -0.0196 0.0035  -0.0889 -0.9947 -0.0142
              -0.1529 -0.0177 0.0718  -0.0238 -0.1683 -0.0267 0.9696  0.0078  0.0332  -0.0161 -0.0002
              -0.7306 -0.0679 -0.0102 -0.2215 0.4024  -0.0260 -0.0338 -0.0594 -0.4903 0.0563  -0.0409
              0.0967  0.1213  -0.1875 -0.2387 0.6970  0.3151  0.1380  -0.1420 0.5007  -0.0706 0.0616
              -0.5806 -0.2342 0.2351  0.0380  -0.2662 0.0275  -0.1811 -0.0234 0.6667  -0.0426 -0.0082
              -0.0411 -0.0116 0.0125  0.0057  -0.0288 -0.0231 -0.0115 0.0056  -0.0463 -0.0078 0.9971
              0.0606  -0.0058 0.0687  0.0006  -0.0944 -0.1381 -0.0064 -0.9807 -0.0414 0.0051  -0.0008
              0.2317  -0.0822 0.7232  -0.5336 0.1465  -0.3172 -0.0163 0.0950  0.0187  -0.0005 -0.0004
```

Figura 17. Autovalores y autovectores obtenidos en "C++".

Referencias

Kaya, H. (s.f.). *UCI.* Obtenido de https://archive.ics.uci.edu/ml/datasets/Gas+Turbine+CO+and+NOx+Emission+Data+Set

netlib. (s.f.). *LAPACK.* Obtenido de http://www.netlib.org/lapack/explore-html/d1/d7e/group__double_g_esing_ga84fdf22a62b12ff364621e4713ce02f2.html

ULPGC. (s.f.). *BLAS.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/411998/mod_resource/content/9/4%20BLAS.pdf

ULPGC. (s.f.). *LAPACK.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/412001/mod_resource/content/7/5%20LAPACK.pdf

ULPGC. (s.f.). *OpenMP.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/412007/mod_resource/content/0/8.%20Introducci%C3%B3n%20a%20la%20Programaci%C3%B3n%20OpenMP.pdf