Métodos Numéricos Avanzados

Trabajo Práctico 1

Segundo Cuatrimestre 2017

Facial recognition

Autores:

José Carlos Noriega Defferrarri - 51231

Agustin Scigliano - 51277

Tomás de Lucca - 52051

Juan Marcos Bellini - 52056

Keywords:

Principal Component Analysis, Kernel Principal Component Analysis, PCA, KPCA, Eigenvalues, Eigenvectors, Eigenfaces, Linear Algebra

21 de Septiembre de 2017

Índice

Abstract

El reconocimiento facial tiene muchas aplicaciones. Existen usos concretos tanto pasos fronterizos[1], como en métodos de acceso mediante datos biométricos a dispositivos móviles[2]. El presente informe documenta el uso de PCA y Kernal PCA para el reconocimiento facial.

1. Introducción

Dado que el reconocimiento facial es una herramienta útil para agilizar procesos, uno debe entender cuáles son los métodos para lograr esto, en un tiempo aceptable. También, que tan preciso es, para determinar que uso darle.

El informe trata de una implementación de reconocimiento facial mediante dos algoritmos distintos de pre-procesamiento de datos para comparar su eficiencia en cuanto a tiempo y precisión.

Los métodos previamente mencionados son:

- Principal Component Analysis (**PCA**)[3]
- Kernel Principal Component Analysis (**KPCA**)[4]

A su vez, considerando el importante rol que tienen los autovectores y autovalores para el desarrollo, se implementaron dos métodos de descomposición QR[5] para su mejor análisis y entendimiento del tema.

2. Metodología

2.1. Implementación

El desarrollo fue realizado en su totalidad en el lenguaje de programación *python* debido a su facilidad y simplicidad de uso a la hora de realizar operaciones con matrices.

El cálculo de autovectores se realiza por medio de la descomposición QR y posterior utilización de un método global de aproximación iterativo.

2.2. Cálculo de autovectores y autovalores

2.2.1. Descomposición QR

Se implementaron dos métodos para obtener la descomposición QR de matrices simétricas: Gram-Schmidt[6] y Householder[7]. Posteriormente se procedió a realizar una

[1] https://www.theverge.com/2017/5/9/15591648/airport-facial-recognition-customs-tsa-biometric-exit

[2] https://en.wikipedia.org/wiki/Face_ID

[3] https://en.wikipedia.org/wiki/Principal_component_analysis

[4] https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

[5] https://en.wikipedia.org/wiki/QR_decomposition

[6] https://en.wikipedia.org/wiki/QR_decomposition#Using_the_Gram.E2.80.93Schmidt_process

[7] https://en.wikipedia.org/wiki/QR_decomposition#Using_Householder_reflections

comparación de la performance entre ambas técnicas. Los métodos reciben la matriz a descomponer y devuelven dos matrices: Q, ortogonal, y R triangular superior, de manera tal que $Q * R$ recrean la matriz inicial.

Gram-Schmidt

El algoritmo de Gram Schmidt fue implementado tomando las columnas de la matriz A a descomponer, y aplicando el método de Gram-Schmidt. Matemáticamente, el método se resume en las siguientes ecuaciones:

$$\vec{q_1} = \frac{\vec{a}_1}{||\vec{a}_1||} \tag{1}$$

$$\vec{q_k} = \vec{a}_k - \sum_{l=1}^{k-1} (\vec{a}_k^T \vec{q_l}) \vec{q_l} \text{ para } k \in \{2, 3, .., n\} \text{ y con } \vec{q_k} = \frac{\vec{q_k}}{||\vec{q_k}||} \tag{2}$$

Posteriormente se calcula $R = Q^T A$.

```
1  def _gram_schmidt(matrix):
2
3      m, n = matrix.shape
4      q = np.zeros((m, n))
5      r = np.zeros((n, n))
6      for j in range(n):
7          v = matrix[:, j]
8          for i in range(j):
9              r[i, j] = np.matmul(q[:, i].T, matrix[:, j])
10             v = v.squeeze() - np.dot(r[i, j], q[:, i])
11         r[j, j] = np.linalg.norm(v)
12         q[:, j] = np.divide(v, r[j, j]).squeeze()
13     return q, r
```

Algoritmo 1: Cálculo descomposición QR por Gram-Schmidt

Reflexiones de Householder

Consiste en transformar progresivamente los vectores columna de la matriz A a descomponer en una matriz triangular superior. Se multiplica la matriz A por la matriz Q (de Householder), que se obtiene al elegir la columna de la matriz original. De esta forma se busca iterativamente las n matrices Q_i de Householder tales que multiplicadas por la matriz A resulta una triangular superior. De esta forma:

$$Q = Q_1 Q_2 ... Q_n \tag{3}$$

$$R = Q_n ... Q_2 Q_1 A \tag{4}$$

$$A = Q^T R \tag{5}$$

```
def _householder(matrix):

    m, n = matrix.shape
    q = np.identity(m)
    r = matrix.copy()
    for i in range(0, m - 1):
        v = r[i:m, i]
        s = -np.sign(v[0]).item()
        norm = np.linalg.norm(v)
        u = (r[i, i] - (norm * s)).item()
        v = np.divide(v, u)
        v[0] = 1
        tm = np.matmul(v, v.T) * (-s * u) / norm
        r[i:, :] = np.subtract(r[i:m, :], np.matmul(tm, r[i:m, :]))
        q[:, i:] = q[:, i:] - np.matmul(q[:, i:], tm)
    return q, r
```

Algoritmo 2: Cálculo de descomposición QR por el método de Householder

2.2.2. Cálculo de autovectores y autovalores

Para el calculo de los autovalores y autovectores se utilizó un método global de aproximación, empleando las técnicas de descomposición QR descriptas anteriormente. Se siguió el proceso descripto en un documento[8].

Es relevante aclarar que, debido a la naturaleza del problema, se necesitará calcular autovectores y autovalores de una matriz simétrica. Esto es así ya que en el problema puntual, la matriz a trabajar, es el resultado de multiplicar una matriz dada por su traspuesta. Debido a esto, el algoritmo se ve simplificado.

El método funciona de la siguiente forma: Se calcula la descomposición QR de la matriz A y luego se obtiene $A_1 = RQ$. Iterativamente se vuelve a calcular la descomposición QR de A_1 y se obtiene A_2, análogamente. Eventualmente se convergerá a una matriz diagonal A_n (es diagonal porque A es simétrica) en la cual cada elemento de la diagonal representa un autovalor de A. De la misma forma, con las matrices Q_i intermedias que se van obteniendo, se puede obtener la matriz de autovectores $S = QQ_1Q_2...Q_n$ (esto también sucede únicamente porque la matriz A es simétrica) en la cual cada columna representa el autovector respectivo a cada autovalor obtenido previamente.

```
def _symmetric_eig(matrix, method=_householder, iterations=50,
    tolerance=1e-4):

    a = np.matrix(matrix, dtype=np.float64)

    q, r = method(a)
    a = np.matmul(r, q)
```

[8] http://www-users.math.umn.edu/~olver/aims_/qr.pdf

```
    s = q

    for i in range(iterations):
        q, r = method(a)
        a = np.matmul(r, q)
        s = np.matmul(s, q)
        if np.allclose(a, np.diagflat(np.diag(a)), atol=tolerance):
            break

    eigenvalues = np.diag(a)
    return eigenvalues, s
```

Algoritmo 3: Cálculo de autovalores y autovectores

2.3. Principal Component Analysis

El principal objetivo de este algoritmo es la reducción de la información para su posterior procesamiento. La idea es poder reducir el espacio de variables posiblemente relacionadas a variables independientes. De esta manera, se logra tener la información mas relevante en un espacio acotado.

Dado que PCA construye una transformación lineal de los datos, y para el reconocimiento facial se tiene en las imágenes un modelo de datos lineal, es muy útil su aplicación.

A continuación se explicará como, mediante la búsqueda de los autovectores de la matriz de covarianza, formada con las imágenes de entrenamiento, se logra construir un espacio vectorial que es capaz de describir a cualquier otra imagen como punto de entrada.

2.3.1. Cálculo de autocaras

Una imagen puede ser representada con una matriz de dimensión $N \times R$ (2D). Lo primero que se debe hacer es transformar dicha matriz en un vector de $N \times R$ elementos. Aplicando esta operación a cada imagen, se obtiene el conjunto $\{\Gamma_1, \Gamma_2, ..., \Gamma_M\}$.

Siguiente a esto, se debe calcular la *cara promedio*. Esto se hace de la siguiente manera.

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} (\Gamma_i) \tag{6}$$

Una vez obtenida la cara promedio, se debe restar a cada vector imagen, obteniendo los siguientes vectores.

$$\Phi_i = \Gamma_i - \Psi \tag{7}$$

Con dichos vectores, se arma una matriz A, que es una concatenación de los Φ.

$$A = \left[\begin{array}{cccc} \Phi_0, & \Phi_1, & \dots & , \Phi_M \end{array} \right] \tag{8}$$

Para aplicar PCA se deben encontrar los autovectores de la matriz $A \times A^T$.

$$C = \frac{1}{M} \sum_{i=1}^{M} (\Phi_i {\Phi_i}^T) = AA^T \tag{9}$$

$$AA^T \vec{v_i} = \mu_i \vec{v_i} \tag{10}$$

Esta matriz es de $(N \times R) \times (N \times R)$, con lo que calcular autovectores, resulta muy costoso computacionalmente. Es así que se aplican ciertas operaciones para lograr calcular autovectores y autovalores a una matriz de $M \times M$ con $M \ll N \times R$

Para ello, se multiplica por A^T en ambos lados de la ecuación (10) para obtener

$$A^T AA^T \vec{v_i} = \mu_i A^T \vec{v_i} \tag{11}$$

De ahí se puede ver que $A^T \times \vec{v_i}$ es autovector de $A^T \times A$. Se define así

$$\vec{u_i} = A^T \vec{v_i} \tag{12}$$

Tomando la definición de autovector y autovalor

$$A^T A\vec{u_i} = \mu_i \vec{u_i} \tag{13}$$

Multiplicando por A de ambos lados en (13)

$$AA^T A\vec{u_i} = \mu_i A\vec{u_i} \tag{14}$$

Se llega a que Au_i es autovector de AA^T. Se define como w.

$$AA^T A\vec{u_i} = \mu_i A\vec{u_i} = \mu_i \vec{w_i} \Rightarrow \vec{w_i} = A\vec{u_i} \tag{15}$$

De esta manera se calculan los autovectores que necesarios utilizando una matriz mas chica, haciendo los cálculos mas livianos computacionalmente. Estos nuevos autovectores son conocidos como autocaras debido a que al ser modificados para que tengan la dimensión original de las imágenes, al ser representadas, el dibujo es muy parecido al de una cara.

Figura 1: Autocara de ejemplo

2.3.2. Autocaras necesarias

Cuando se calcularon los autovectores y autovalores de la matriz reducida, si se ordenan los autovalores de manera decreciente, junto a sus respectivos autovectores, se puede ver que el autovector asociado al autovalor mas grande representa la mayor varianza en la imagen. De esta manera, se puede tomar un porcentaje de los autovectores mas relevante. Esto se llama energía. El proceso es de la siguiente manera:

1. Ordenar los autovalores de forma decreciente junto con sus respectivos autovectores.
2. Sumar los autovectores. A esta suma se la llama energía.
3. Calcular el porcentaje de energía necesario.
4. Tomar los primeros K autovectores cuyos autovalores asociados suman el porcentaje de energía.

2.3.3. Proyección del conjunto de entrenamiento

Para poder representar las imágenes en términos de las autocaras, se deben proyectar los vectores Φ en los vectores $\{\vec{v_1}, \vec{v_2}, \ldots, \vec{v_M}\}$ (las autocaras). Estas proyecciones serán luego usadas en la clasificación, para el posterior reconocimiento. A continuación se puede ver el cálculo que se debe hacer.

$$\Omega = A \left[\begin{array}{cccc} \vec{w_0}, & \vec{w_1}, & \ldots & , \vec{w_M} \end{array} \right] = \left[\begin{array}{cccc} \vec{\omega_0}, & \vec{\omega_1}, & \ldots & , \vec{\omega_M} \end{array} \right] \tag{16}$$

O sea, a través de una única operación matricial se puede obtener, en una matriz, las proyecciones. Recordar que la matriz A fue definida en (8), y que $[\vec{v_1}, \vec{v_2}, \ldots, \vec{v_M}]$ es una matriz que contiene a las autocaras como columnas.

Finalmente, al intentar reconocer una imagen no utilizada en el entrenamiento (pasos anteriores), la misma debe ser sometida al mismo procedimiento que las demás imágenes (restarle la media y luego ser proyectada en las autocaras).

2.4. Kernel Principal Component Analysis

El objetivo de KPCA es proyectar datos que no puedan ser separados linealmente a una dimensión superior, en la que sí lo sean. Dado un mapeo no lineal, se puede mapear un espacio $\Re^n$, a un espacio F

$$\begin{aligned} \Phi : x &\mapsto \Phi(x) \\ R^n &\to F \end{aligned} \tag{17}$$

Una de las motivaciones de KPCA es aplicar PCA sobre este nuevo espacio F. Pero debido a que es difícil centralizar los datos en F, y computacionalmente costoso o hasta imposible calcular los autovectores de la matriz de covarianza C en F, se procede a utilizar "kernel tricks".

Siendo $A = [\Phi_0, \Phi_1, \ldots, \Phi_M]$ se define la función polinomial kernel como:

$$k(x, y) = ((x \cdot y)/m + 1)^d \tag{18}$$

De esta manera, se obtiene R_{i_j} como

$$\begin{aligned} R_{i_j} &= k(x_i, x_j) \\ &= ((\Phi_i \cdot \Phi_j)/M + 1)^d \end{aligned} \tag{19}$$

siendo M la cantidad de imágenes a procesar.

Luego se debe centrar R haciendo el siguiente cálculo

$$R' = R - 1_M R - R 1_M + 1_M R 1_M \tag{20}$$

siendo $(1_M)_{ij} = 1/M$.

Luego se deben obtener los autovectores $u_1, u_2, \ldots$ y los autovalores $\lambda_1, \lambda_2, \ldots$ de R' para poder calcular los autovectores $w_1, w_2, \ldots$ de la matriz de covarianza C en F mediante la proyección:

$$w_i = 1/\sqrt{\lambda_i} A u_i \tag{21}$$

Finalmente se aplica PCA como se vio en la sección anterior.

2.5. Clasificación

Para la clasificación se utilizó Support Vector Machine (SVM). En particular, linear SVM. Los motivos que llevaron a usar este algoritmo fue que la eficiencia del mismo, el fácil uso, y la buena documentación. No se ha hecho un análisis profundo sobre el tema, ya que el foco está en el pre-procesamiento de los datos, y con tener un clasificador confiable y respaldado por la comunidad, es suficiente.

3. Resultados y conclusiones

Householder vs. Gram-Schmidt

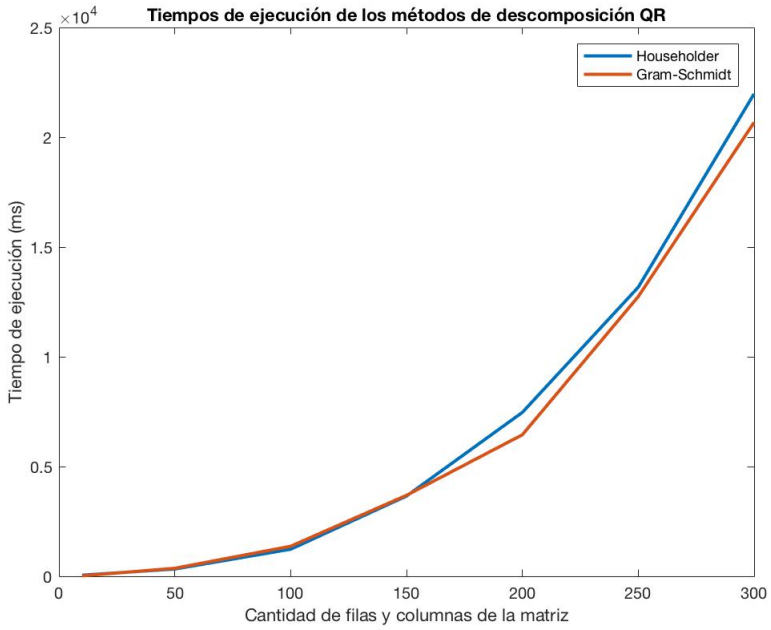

Figura 2: Resultados de la performace de ambas técnicas de descomposición QR

Las pruebas fueron realizadas con matrices simétricas generadas aleatoriamente, y se tomo el promedio de las distintas muestras por cada tamaño de matriz distinto.

Los resultados obtenidos indicaron que la performance en promedio de ambos métodos es similar en términos del tiempo de ejecución del algoritmo. Sin embargo, se notó empíricamente que las aproximaciones obtenidas en el mismo tiempo por el método de Householder contenían menor error que las obtenidas mediante el algoritmo de Gram-Schmidt.

Debido a que el algoritmo de Householder tiene una mayor estabilidad numérica, es decir, que los errores debidos a las aproximaciones se atenúan a medida en que el algoritmo progresa, es computacionalmente más costoso calcularlo pero a la vez arroja mejores resultados. El algoritmo de Gram-Schmidt es numéricamente inestable y, por lo tanto, la precisión del mismo suele ser inferior, pero ofrece mejores tiempos de cálculo.

PCA vs. KPCA

La siguiente prueba fue realizada utilizando la base de datos del Olivetti Research Laboratory, la cual se puede encontrar en el siguiente enlace: `http://www.cl.cam.`

ac.uk/research/dtg/attarchive/facedatabase.html. Se han tomado 6 imágenes por sujeto para entrenamiento, y las 4 restantes, para *testeo*.

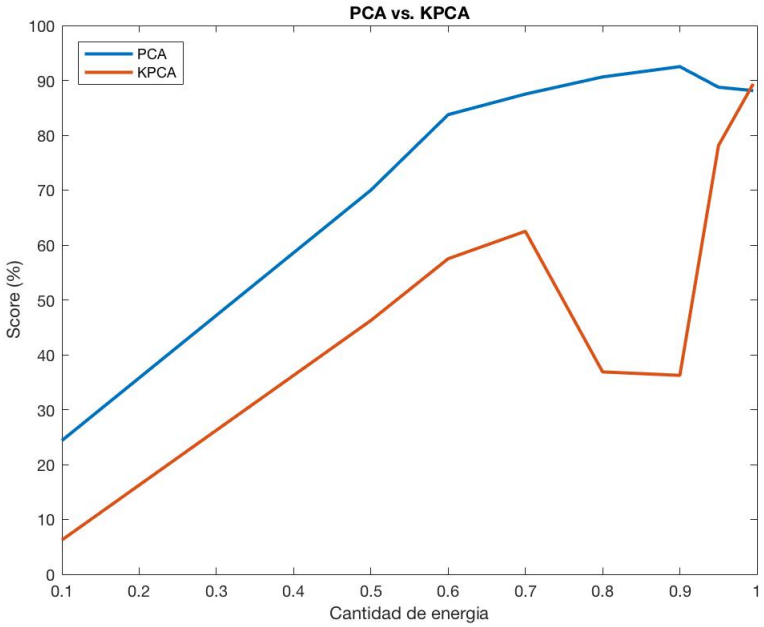

Figura 3: PCA vs. KPCA

Como se puede apreciar en la figura 3, se realizó la comparación de los dos métodos de reconocimiento facial implementados. Se utilizaron distintos niveles de energía y se calculo el grado de precisión del sistema. Los resultados empíricos luego de muchas pasadas nos muestran que se obtuvieron mejores resultados con PCA que con KPCA para todos los niveles de energía, salvo el último.

Sorpresivamente en los casos con energía de 0.8 y 0.9 los valores dieron sistemáticamente muy distintos al resto, y al comportamiento esperado de la función. Se desconoce con exactitud el motivo de este resultado, aunque se estima que puede ser algún tipo de ruido en el cálculo de autovectores.

Distintos grados del polinomio de Kernel PCA

La siguiente prueba fue realizada utilizando la base de datos del Olivetti Research Laboratory, la cual se puede encontrar en el siguiente enlace: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html. Se han tomado 6 imágenes por sujeto para entrenamiento, y las 4 restantes, para *testeo*.

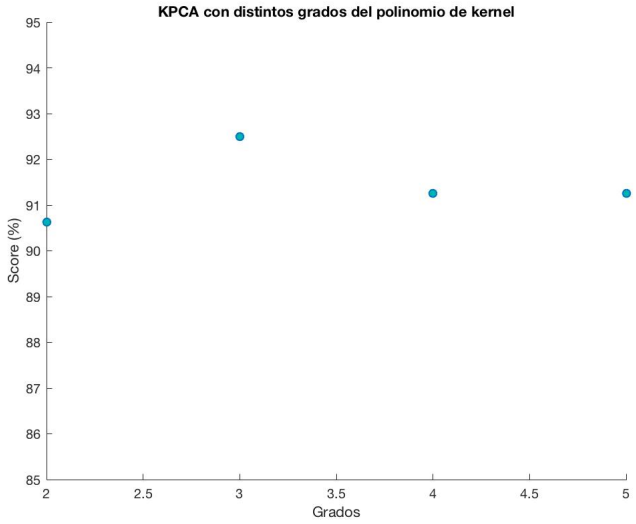

Figura 4: KPCA con distintos grados del polinomio de kernel

Como se puede observar en la figura 4, se han realizado comparaciones de Kernel PCA utilizando distintos grados en la funcional polinomial. Los mejores resultados se obtienen con el polinomio de grado 3. A medida que se aumenta el grado la precisión del sistema decrece debido a que los datos son llevados a valores demasiado altos.

4. Bibliografía

- Wikipedia: Autovalores y autovectores https://es.wikipedia.org/wiki/Vector_propio_y_valor_propio
- Wikipedia: Eigenface https://en.wikipedia.org/wiki/Eigenface
- Wikipedia: Análisis de componentes principales https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales
- Wikipedia: Kernel principal component analysis https://en.wikipedia.org/wiki/Kernel_principal_component_analysis
- Wikipedia: QR decomposition https://en.wikipedia.org/wiki/QR_decomposition
- Wikipedia: Support vector machine https://en.wikipedia.org/wiki/Support_vector_machine
- Youtube https://www.youtube.com/watch?v=SaEmG4wcFfg
- Gram Schmidt in 9 Lines of MATLAB http://web.mit.edu/18.06/www/Essays/gramschmidtmat.pdf
- Householder QR http://www.cs.cornell.edu/~bindel/class/cs6210-f09/lec18.pdf
- Orthogonal Bases and the QR Algorithm http://www-users.math.umn.edu/~olver/aims_/qr.pdf
- Face Recognition using Principle Component Analysis http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaface.pdf
- Face Recognition base on KPCA with polynomial Kernels
- Apuntes de cátedra
- Matthew Turk and Alex Pentland. Eigenfaces for recognition.
- M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In Proceedings.
- Kwang In Kim, Keechul Jung, and Hang Joon Kim. Face recognition using kernel principal component analysis.
- Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem.
- Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis, pages 583–588.
- K. R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms.
- Bernhard Schölkopf and Alexander J Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond.