Reconocimiento facial con PCA y KPCA

Métodos Numéricos Avanzados
Instituto Tecnológico de Buenos Aires

Balaguer, Pedro 55795
Benítez, Julián 56283
Garrigó, Mariano 54393
Perazzo, Matías 55024
Saqués, M. Alejo 56047

Resumen

Se han empleado técnicas matriciales, tales como *Principal Component Analysis* y su variante, *Kernel Principal component Analysis* para la clasificación de caras en sus respectivas clases, teniendo como referencia *sets* de entrenamiento.

Asimismo, se han implementado algoritmos para el hallado de los autovectores y autovalores de una matriz, y se han explorado numerosos caminos para la descomposición QR, como así también para otras transformaciones que sirven de soporte para la implementación de un algoritmo QR mejorado.

Tras minuciosos análisis comparativos entre las dos metodologías implementadas, se pudo concluir que **KPCA** exhibe un comportamiento superior frente a **PCA**, dado que la proporción de aciertos del primer método demostró ser consistentemente superior a la del último.

Por último, al realizar pruebas con *kernels* polinomiales de diferentes grados, no ha podido afirmarse que a grados superiores la precisión de la clasificación sea, de mismo modo, superior.

Palabras clave: Descomposición QR, Transformación de Householder, Matriz de Hessemberg, Algoritmo QR, Corrimiento de Wilkinson, Matriz de Covarianza, *Principal Component Analysis*, *Kernel*, *Support Vector Machines*.

1. Metodologías para reconocimiento de caras

A continuación, se exhibirán las metodologías utilizadas para *software* de reconocimiento de caras, utilizando técnicas requeridas por la Cátedra.

1.1. PCA

Para la implementación de reconocimiento de caras utilizando **PCA**, el equipo se ha basado en la técnica de Saha y Bhattacharjee (2013) [1].

Sea N la cantidad de individuos presentes en la base de datos, y sea M la cantidad de imágenes de cada uno de ellos que se usarán a los efectos de *entrenar* el *software* de reconocimiento. Cada imagen tiene una dimensión $X \times Y$. Luego, se debe construir una matriz $T \in \mathbb{N}^{MN \times XY}$ que contenga todas las muestras a analizar. Luego, a la matriz T se le resta fila por fila la *cara promedio* M, es decir, la media de las filas de T. Esta matriz $C = T - M$ puede contener valores negativos, por lo que $C \in \mathbb{Z}^{MN \times XY}$.

El siguiente paso en el método de **PCA** es el hallado de los autovectores de la matriz de covarianza $S = C^T C$. Dichos autovectores son también referidos en la bibliografía por el nombre de *autocaras o eigenfaces*. Es preciso observar que el producto $C^T C \in \mathbb{Z}^{XY \times XY}$, por lo cual, dado que $X \times Y$ es el número de píxeles de

las imágenes, es de esperar que la matriz de covarianza S sea de una dimensión excesivamente grande. Por ende, el objetivo es calcular las *eigenfaces* de una forma computacionalmente más económica.

Nótese que la dimensión de la matriz $\widetilde{S} = CC^T$ ($\widetilde{S} \in \mathbb{Z}^{MN \times MN}$) está intrínsecamente asociada al tamaño de la muestra, ya sea del número de individuos y/o del de caras por individuo. Estos valores serán, en la mayoría de los casos, de una magnitud muy inferior a la cantidad de píxeles de las imágenes y, además, las posibilidades de manipularlos son mucho más amplias. A modo de ejemplo, las imágenes de la base de datos de prueba sugerida por la Cátedra [2] tienen una dimensión de 92×112, por lo que la matriz de covarianza $S \in \mathbb{Z}^{10304 \times 10304}$. Por el contrario, $\widetilde{S} \in \mathbb{Z}^{400 \times 400}$, en el caso de tomar todas las fotos disponibles de cada individuo.

Se puede mostrar que hay un procedimiento para calcular las *eigenfaces* de S a partir de los autovectores de $\widetilde{S}$:

$$Svi = C^T C v_i = \lambda_i v_i \tag{1}$$

Como se ha visto, S es de una dimensión potencialmente muy grande, por ende, se toman los autovectores de la matriz $\widetilde{S}$:

$$\widetilde{S} u_i = CC^T u_i = \lambda_i u_i \tag{2}$$

Multiplicando por izquierda a (2) por C^T se tiene que:

$$C^T C C^T u_i = \lambda_i C^T u_i \tag{3}$$

Luego, se puede renombrar:

$$C^T u_i = v_i \tag{4}$$

Por lo que si u_i es un autovector de $\widetilde{S}$, empleando (4) se puede obtener el i-ésimo autovector de S, es decir, de la matriz de covarianza.

Así pues, en este punto se tiene una colección de vectores $V = [v_1 \quad \dots \quad v_{MN}]$. El interés es ahora proyectar las imágenes de la base de datos sobre estas *eigenfaces*, para formar, por cada k-ésima imagen, un vector $\Omega_k = [\omega_1 \quad \dots \quad \omega_{MN}]$, donde los ω_i representan el peso de cada *eigenface* con respecto a dicha imagen:

$$\omega_i = v_i^T P_k^T \tag{5}$$

Siendo P_k la k-ésima fila de C. Luego, una opción es promediar los M Ω_{k_s} de cada individuo de la base de datos, así logrando un valor medio Ω_{M_p} por cada ejemplar $1 \leq p \leq N$, obteniendo un vector que defina la *clase* de un individuo en particular. Otra posibilidad es no realizar ningún promedio alguno, entregando a la función clasificadora tantos vectores Ω como cantidad de imágenes por individuo haya. En la sección de resultados se analizarán las dos posibilidades en detalle.

Luego, para una imagen de entrada Z a clasificar (dada en forma de columna), se debe computar el peso de las *eigenfaces* con respecto a la misma como se describe en (5), teniendo la precaución de *centralizar* la entrada, es decir, hacer la resta $\widetilde{Z} = Z - M^T$. Luego, reemplazando P_k por $\widetilde{Z}$ en dicha ecuación, se calcula el vector Ω_Z.

Finalmente, se toma la distancia euclídea $\epsilon_p = \|\Omega_Z - \Omega_{M_p}\|$ − considerando el caso en el que los vectores Ω se promedien − para cada $1 \leq p \leq N$, y se dice que el p que corresponda al ϵ_p mínimo determina la clase a la que pertenece la imagen de entrada.

Existe otra técnica para realizar el último cálculo, por medio del uso de *Support Vector Machines*. En la implementación realizada, se ha utilizado la librería `Sklearn` para dicho propósito. Su implementación de SVM espera, para el *set* de entrenamiento, un arreglo de vectores que hagan las veces de los *individuos*, y un arreglo de enteros que especifique a que clase pertenece cada uno de ellos. En este caso,

el primer arreglo contendrá los Ω_{M_p}. La entrada del método `predict` será el Ω_Z a clasificar.

1.2. Kernels y KPCA

1.2.1. Funciones *Kernel*

Los métodos de *kernel* son métodos de reconocimiento de patrones que permiten trabajar en altas dimensiones de forma implícita, es decir sin nunca computar coordenadas en dicho espacio. Utilizar estos métodos es computacionalmente mucho más eficiente a la hora de trabajar en dimensiones altas, y permite obtener datos analizables de forma lineal utilizando los denominados *hiperplanos*. Este tipo de enfoque es llamado *kernel trick* o truco de *kernel* [3].

Las funciones de *kernel* son definidas de la siguiente manera:

$$k : X \times X \to \mathbb{R} \tag{1}$$

Para facilitar las operaciones, se puede notar:

$$\phi : \mathbb{R}^d \to \mathbb{R}^k, k > d \tag{2}$$

Siendo ϕ la función que *mapea* los vectores x a una dimensión superior utilizando combinaciones no lineales de las componentes de x. A modo de ejemplo, supóngase a $x \in \mathbb{R}^d$:

$$x = [x_1 \quad x_2 \quad \dots \quad x_d]^T$$
$$\phi(x) = \begin{bmatrix} x_1 & x_2 & x_1x_2 & x_1^2 & x_1x_2^3 & \dots \end{bmatrix}^T$$

Luego, llámese *función kernel* al producto interno de estas proyecciones de x:

$$\kappa(x_i, x_j) = \phi(x_i)\phi(x_j)^T \tag{3}$$

Existen numerosas funciones de *kernel* conocidas. Algunos ejemplos son [4]:

- *Kernel* polinómico:
  $\kappa(x, x') = (1 + xx')^p$
- *Kernel* gaussiano:
  $\kappa(x, x') = \exp^{-V\|x-x'\|^2}$
- *Kernel* de tangente hiperbólica:
  $\kappa(x, x') = \tanh(xx' + \delta)$

Cabe aclarar que dichas funciones de *kernel* son un listado muy reducido de las existentes. Hallar la función que más se adecue al problema a resolver puede ser una tarea compleja.

1.2.2. *Kernel* PCA

Asumiendo que los datos son difícilmente linealizables, **KPCA** ofrece una gran ventaja frente a **PCA** ya que, como se explicó anteriormente, permitirá linealizar la información a través de una transformación a una dimensión superior. Para comenzar, una función de *kernel* debe ser aplicada sobre los datos.

Considérese la siguiente expresión $\kappa(x_i, x_j) = \phi(x_i)\phi(x_j)^T = K_{ij}$.

Como no se tiene garantizado que los datos en el *kernel* estén centrados, se recurrirá a centrarlos según la siguiente ecuación:

$$\Phi_i := \Phi_i - \frac{1}{N}\sum_k \Phi_k \tag{4}$$

Luego, la matriz de *kernel* centrada resulta:

$$K_C = K - 1_{\frac{1}{N}}\mathbf{K} - \mathbf{K}1_{\frac{1}{N}} + 1_{\frac{1}{N}}\mathbf{K}1_{\frac{1}{N}} \tag{5}$$

Donde $1_{\frac{1}{N}} \in \mathbb{R}^{N\times N}$ una matriz con todos valores $\frac{1}{N}$ ($\mathbf{K} \in \mathbb{R}^{N\times N}$).

Luego, de manera idéntica a **PCA**, se buscan hallar los autovalores y autovectores de una matriz, en este caso, de K_C. Esto puede considerarse equivalente a aplicar la técnica de **PCA** en la dimensión superior $\mathbb{R}^K$.

El ordenamiento de los autovalores y sus respectivos autovectores en orden decreciente permite seleccionar las componentes más significativas del análisis, para luego proyectar las imágenes sobre los mismos y generar las *eigenfaces*. Un similar procedimiento de reducción puede realizarse en **PCA**.

2. Algoritmos de soporte

A los efectos de implementar la metodología expuesta, es preciso contar con un método para el cálculo de los autovalores y autovectores de una matriz. A continuación, se expondrán los detalles de dicho algoritmo, el **algoritmo QR**, como así también los detalles de otras implementaciones necesarias para que éste funcione.

2.1. Algoritmo QR

Sea $A \in \mathbb{R}^{N \times N}$ la matriz cuyos autovalores quieren determinarse. Sea $A_k =: Q_k R_k$ la descomposición QR de la matriz A_k, donde Q_k es ortogonal y R_k es triangular superior. Entonces el algoritmo QR se define de la siguiente forma:

$$A_0 = A \tag{1}$$
$$A_{k+1} = R_k Q_k \tag{2}$$

Nótese que, como los Q_k son ortogonales, se cumple la siguiente igualdad:

$$\begin{aligned} A_{k+1} &= R_k Q_k \\ &= Q_k^{-1} Q_k R_k Q_k \\ &= Q_k^{-1} A_k Q_k \end{aligned}$$

Por ende, los A_k son similares entre sí $\forall k \geq 0$.

Siguiendo este procedimiento, para $k \to$ ínf, la matriz A_k converge a una forma triangular, lo que implica que sus autovalores se encuentran sobre la diagonal.

Un aspecto interesante de dicha convergencia es que sucede de forma **ordenada**, i.e. primero converge el valor $A_{N,N}$, luego $A_{N-1,N-1}$ y así sucesivamente. Por ende, la primer optimización que puede realizarse es la siguiente:

Sea i la i-ésima iteración del algoritmo. Luego, si $|A_{i+1_{j,j}} - A_{i_{j,j}}| \leq \epsilon$ con $\epsilon \to 0$, entonces puede estimarse que el autovalor $A_{j,j}$ ha convergido. Luego, la fila y la columna j-ésimas pueden descartarse en las sucesivas iteraciones, para proceder el algoritmo con una matriz en $\mathbb{R}^{j-1 \times j-1}$. De esta forma, la complejidad computacional del algoritmo se ve reducida.

Otras posibles optimizaciones del algoritmo versan sobre el aceleramiento de su convergencia. Por ejemplo, es posible llevar la matriz A_0 a la forma de Hessemberg $H = Q^{-1} A_0 Q$, siendo H una matriz con 0_s por debajo de la subdiagonal. Considerando que mediante el algoritmo QR se converge a una matriz triangular, establecer $A_0 = H$ acelera la convergencia de los autovalores.

Por último, a los efectos de acelerar la convergencia, se pueden introducir corrimientos o *shifts* κ_k en cada iteración, de la siguiente forma:

$$A_k - \kappa_k I =: Q_k R_k \tag{3}$$
$$A_{k+1} = R_k Q_k + \kappa_k I \tag{4}$$

Introduciendo los corrimientos, se preserva la similaridad entre todos los A_k. Por un lado, tenemos que:

$$Q_k^{-1}(A_k - \kappa_k I) Q_k = Q_k^{-1} Q_k R_k Q_k \tag{5}$$
$$= R_k Q_k \tag{6}$$

Luego, reemplazando $R_k Q_k$ en (7):

$$A_{k+1} = Q_k^{-1}(A_k - \kappa_k I) Q_k + \kappa_k I \tag{7}$$
$$= Q_k^{-1} A_k Q_k - \kappa_k Q_k^{-1} Q_k + \kappa_k I \tag{8}$$
$$= Q_k^{-1} A_k Q_k \tag{9}$$

Por lo que $A_{k+1} \sim A_k$. Idealmente, en cada paso $\kappa_k = \lambda_i$, siendo λ_i el autovalor al que converge el valor inferior derecho de la diagonal. Como precisamente el problema que se intenta resolver es obtener los autovalores de una matriz, se deberán utilizar heurísticas para aproximar dichos valores en cada iteración. Una heurística posible es la *trivial*: tomar exactamente el extremo inferior derecho de la diagonal como κ_k. Existen otras heurísticas, tales como los corrimientos de Wilkinson, que buscan aproximar uno de los autovalores de la matriz de 2×2 de la esquina inferior derecha.

Para evaluar el comportamiento del algoritmo frente a diferentes tipos de corrimiento, se han tomado mediciones del tiempo de ejecución del algoritmo QR utilizando diferentes heurísticas para los *shifts*. Para cada una de las dimensiones del cuadro siguiente, se han generado matrices aleatorias cuyos autovalores − por medio de una transformación de similaridad − sean exactamente los de $\Lambda = [1 \quad \dots \quad dim]$. En 2.1 se exhiben los resultados obtenidos.

Dim.	Sin *shifts*	Wilkinson	*Trivial*
10	0.022000	0.013000	0.007000
20	0.093000	1.154000	0.027000
30	0.212000	0.370000	0.071000
40	0.466000	0.262000	0.153000
50	0.859000	0.879000	0.260000
60	1.495000	0.569000	0.385000
70	2.061000	0.844000	0.627000
80	3.141000	4.478000	0.904000
90	4.212000	2.257000	1.543000

Cuadro 1: Comparación en tiempo de ejecución entre diferentes tipos de *shifts*.

Contrariamente a lo que pudo haberse esperado, los corrimientos de Wilkinson implementados no solo demostraron no ser universalmente superiores a los corrimientos *triviales*, sino que en algunos casos tuvieron una *performance* incluso inferior al del algoritmo sin ningún tipo de corrimiento. Uno de los posibles motivos de este resultado podría estar relacionado a que el cómputo de los corrimientos de Wilkinson conlleva una serie de operaciones de punto flotante, tales como raíces cuadradas y divisiones. Por el contrario, los corrimientos *triviales* son de cálculo inmediato.

Considerando los resultados expuestos, se optó finalmente por utilizar los *shifts triviales*.

2.2. Reflectores de Householder

Tal como se ha mencionado en el caso de la matriz de Hessemberg, uno de los intereses que se tiene a menudo es la introducción de 0_s por debajo de la subdiagonal. Unas de las herramientas utilizadas para dicha caso, como así también para la descomposición QR de una matriz, son los reflectores de Householder.

Considérese la siguiente matriz simétrica $P = I - \mu vv^T$. El objetivo es encontrar un μ tal que P sea, además, ortogonal. Para ello, debe cumplirse la igualdad $P^T P = I$ [5].

$$\begin{aligned} P^T P = I &= (I - \mu vv^T)^T (I - \mu vv^T) && (1)\\ &= I - 2\mu vv^T + \mu^2 vv^T vv^T && (2)\\ &= I - 2\mu vv^T + \mu^2 (v^T v) vv^T && (3)\\ &= I + \mu(\mu v^T v - 2) vv^T && (4)\\ 0 &= \mu v^T v - 2 && (5)\\ \mu &= \frac{2}{v^T v} && (6) \end{aligned}$$

Luego, sin perder generalidad, puede fijarse $v^T v = \|v\|_2^2 = 1$, quedando la forma tradicional del reflector de Householder:

$$P = I - 2vv^T \tag{7}$$

Con $\|v\|_2^2 = 1$.

2.3. Descomposición QR

Siendo $A =: A_0$ una matriz cualquiera de $\mathbb{R}^{N \times M}$, se busca una descomposición $A = QR$ tal que Q sea ortogonal y R triangular superior. Utilizando los reflectores de Householder, el interés es encontrar una matriz H de la forma $H = I - 2vv^T$ tal que se cumpla para algún vector x:

$$Hx = \alpha e_1 \tag{1}$$

Con $e1 = [1 \quad 0 \quad \dots \quad 0]^T$.

Aplicando esta limitación y la definición del reflector, se puede llegar a que:

$$v = x + \|x\|_2 e_1 \tag{2}$$
$$c = \frac{2}{\|v\|_2^2} \tag{3}$$
$$H = I - cvv^T \tag{4}$$

Con H ortogonal.

La idea, pues, es generar matrices de Householder H_i tales que $H_1 A_0$ retorne una matriz A_1 con 0_s por debajo de la diagonal en la primera columna, $H_2 A_1$ además agregue 0_s debajo de la diagonal en la segunda columna, y así sucesivamente, hasta lograr un producto $H_{M-1} H_{M-2} \dots H_1 A = R$, tal que R sea triangular superior. Los H_i son de la siguiente forma:

$$H_i = \begin{bmatrix} I_{i-1} & 0 \\ 0 & \widetilde{H}_i \end{bmatrix} \tag{5}$$

Con $1 \leq i \leq M-1$. La matriz $\widetilde{H}_i$ cumple la condición $\widetilde{H}_i A_{i-1_{i:,i:}} = \alpha e_1$, con indexación al estilo de MATLAB. Recuérdese que $A_0 = A$, la matriz de entrada del algoritmo.

Nótese que, como los $\widetilde{H}_i$ son ortogonales, por la construcción en (5) los H_i son ortogonales, luego el producto $H_{M-1} H_{M-2} \dots H_1 = Q^T$ es ortogonal. Por ende, se ha conseguido una matriz R triangular superior, y una Q ortogonal, tal que $QR = A$, por lo que se ha logrado la descomposición requerida.

Se ha evaluado otro método para realizar descomposiciones QR, el cual involucra implementar *rotaciones de Givens* para cada celda por debajo de la diagonal principal. Este método exhibe una mayor complejidad algorítmica, ya que hay que iterar sobre los $O(N^2)$ casilleros por debajo de la diagonal, e introducir un similar orden de rotaciones, efectuando un idéntico número de multiplicaciones matriciales, una operación de orden cúbico. Con los reflectores de Householder, se recorren a lo sumo las $M-1$ columnas de la matriz, introduciendo la cantidad deseada de 0_s por columna en cada iteración.

Dim.	T. Givens	T. Householder
50	0.044000	0.004000
100	0.513000	0.014000
150	2.673000	0.046000
200	8.824000	0.148000
250	26.306000	0.259000
300	60.185000	0.485000

Cuadro 2: Comparación entre implementaciones de descomposición QR (segundos).

En 2.3 se evidencia cómo incluso a dimensiones pequeñas de la matriz, la diferencia en tiempo de ejecución entre las implementaciones es sumamente apreciable.

2.4. Matriz de Hessemberg

En la sección 1.1. se ha comentado acerca de los potenciales beneficios de transformar la matriz de entrada A del algoritmo QR a una forma de Hessemberg superior, es decir con 0_s por debajo de la subdiagonal. A continuación, se abordarán los detalles de dicha transformación.

La reducción a una matriz de Hessemberg procede de manera similar a la descomposición QR por el método de Householder, con la diferencia que, en este caso, se busca generar una matriz $H \sim A$, donde $A \in \mathbb{R}^{N \times M}$ es la entrada del algoritmo.

El objetivo es generar matrices de Householder P_i, con $1 \leq i \leq M-2$, tal que $P_1 A$ anule los elementos de la primera columna por debajo de la **subdiagonal**. Luego, como se busca generar una matriz similar a la entrada, se multiplica por derecha a este resultado por P_1^{-1}.

Procediendo de esta forma hasta $i = M - 2$ se obtiene $H = P_{M-2} \dots P_1 A P_1^{-1} \dots P_{M-2}^{-1}$, con $H \sim A$. Un interesante ejemplo gráfico del procedimiento se puede observar en la referencia a continuación [6].

3. Resultados

A continuación, se discutirán los resultados de la ejecución de ambas implementaciones, tomando como resultado de las mismas la proporción de aciertos por sobre el total de ejemplares.

A la hora de realizar las pruebas, se ha tomado tanto una base de datos de construcción propia como así también la base de datos recomendada por la Cátedra [2]. El objetivo de realizar esto es analizar el comportamiento de los algoritmos frente a bases de datos de diferente calidad.

A modo de aclaración, en cada uno de los casos que se realizaron se ha buscado clasificar imágenes **no incluidas** en el *set* de entrenamiento, a los efectos de evaluar las capacidades de los algoritmos de reconocer imágenes con rasgos distintos a los conocidos.

Notas sobre el procedimiento: Para cada una de las tablas, se están tomando $Img.P/Per$ imágenes de **5 individuos** diferentes. $N.Img.$ representa el número de imagen de cada individuo que se está tomando en cada ocasión. Luego, para cada base de datos analizada, y para cada configuración del par $(N.Img. - Img.P/Per.)$, se están clasificando 5 imágenes, usando un *set* de entrenamiento de un tamaño de $5 * Img.P/Per.$.

3.1. PCA

Tal como se ha planteado en la descripción del método de reconocimiento de caras por *Principal Component Analysis*, una de las opciones es **no** promediar las proyecciones de las imágenes del *set* de entrenamiento con las *eigenfaces*. Como se puede observar en 3, y exceptuando contados casos, no se evidencian diferencias considerables entre el método que realiza los promedios y el que no. Existen algunos casos, tales como el $[N.Img = 9; Img.P/Per = 7]$, en el que, tomando ambas bases de datos, el desempeño del algoritmo que promedia los Ω es considerablemente inferior al del que no realiza dicho promedio.

Sin embargo, el aspecto quizás más destacable que puede observarse en 3 es la diferencia de resultados entre la aplicación de los algoritmos sobre la base de datos sugerida por la Cátedra y la provista por el equipo. A pesar de que se han tomado medidas para minimizar los posibles errores por imágenes poco claras (procurar un ambiente con abundante iluminación, centrar la captura y usar un fondo estandarizado), se puede observar que en cada uno de los casos, los resultados con la primera base de datos son como mínimo igual de precisos que los de la segunda. Posibles soluciones a este hallazgo incluyen tanto ampliar la base de datos propia, como así también procurar un entorno profesional para la captura de imágenes.

Otro aspecto que puede notarse en 3 es que, a medida que aumenta el tamaño del *set* de entrenamiento, la precisión no necesariamente aumenta de igual forma. Esto se puede observar en el caso de la imagen 7: tomando solo 1 muestra por individuo para el entrenamiento, todas las imágenes son correctamente clasificadas. Sin embargo, tomando 3 muestras por individuo, la precisión disminuye en todos los casos.

3.2. Kernel-PCA

En 4 puede evidenciarse como, efectivamente, la técnica del *kernel* arroja resultados mejorados con respecto al **PCA** sin mejoras de tal tipo.

La mejora de los resultados resulta clara en el caso de la base de datos sugerida por la Cátedra, donde prevalecen, salvo contadas excepciones, los aciertos absolutos. Dicha mejoría se

Parámetros		DB. Cambridge		DB. Propia	
N.Img.	Img. P/Per.	C/Prom.	S/Prom.	C/Prom.	S/Prom.
2	1	.8	.8	.8	.8
3	1	1	1	1	1
4	1	.8	.8	.6	.6
7	1	1	1	1	1
4	3	.8	.8	.4	.6
5	3	.6	.6	.6	.8
6	3	.8	.8	.8	.6
7	3	.8	.8	.6	.8
8	3	1	1	.2	.4
8	5	1	1	.6	.4
9	5	.8	.8	.6	.6
8	7	.8	.8	.4	.4
9	7	.4	1	.4	.6

Cuadro 3: **PCA:** Dos bases de datos diferentes

Parámetros		Cambridge		Propia	
N.Img.	Img. P/Per.	Polin. O. 2	Polin. O.4	Polin. O. 2	Polin. O. 4
2	1	.8	**1**	1	1
3	1	1	.8	.8	**1**
4	1	1	1	.6	**.8**
7	1	1	1	1	1
4	3	1	**1**	.6	.6
5	3	.8	.8	1	1
6	3	1	1	.8	.6
7	3	1	1	1	1
8	3	.8	.8	.8	.8
8	5	1	1	.8	.8
9	5	1	1	.8	.8
8	7	.8	**1**	1	1
9	7	1	1	.8	.8

Cuadro 4: **KPCA:** Comparación entre diferentes *kernels* polinomiales

evidencia también en la base de datos provista por el equipo: en casos donde la proporción de aciertos era igual o inferior, en **PCA** a 0,6, en **KPCA** las mismas proporciones nunca son menores a 0,8. Este último ejemplo podría estar indicando el accionar del *kernel*: la mayor cantidad de datos del *set* de entrenamiento pudo ser separada más correctamente en una dimensión elevada.

Como puede notarse, los cambios de precisión utilizando un polinomio de grado 4 no son necesariamente positivos tomando como punto de comparación un **KPCA** que utiliza un *kernel* polinomial de grado 2. En ambas bases de datos pueden observarse tanto aumentos del número de clasificaciones correctas, como así también decrecimiento en las mismas. Como no se pudo corroborar un aumento generalizado de la precisión, podría inferirse que polinomios de grado superior *no necesariamente* garantizan mejores resultados.

4. Conclusión

Se pudo corroborar la mejora que supone el método de *Kernel Principal Component*

Analysis por sobre su versión sin la utilización de funciones de *kernel* para elevar la dimensión de la muestra. Sin embargo, no se ha podido concluir que un *kernel* polinomial de grado mayor a 2 exhiba mejores resultados que uno de dicho grado a la hora de aplicar **PCA**.

Por otro lado, en lo que respecta al algoritmo QR, se han analizado diversos *shifts*, determinándose que los corrimientos de Wilkinson no necesariamente generan una convergencia más rápida frente a un corrimiento trivial.

Referencias

[1] International Journal of Emerging Technology and Advanced Engineering, **Face Recognition Using Eigenfaces**, Rajib Saha, Debotosh Bhattacharjee, http://www.ijetae.com/files/Volume3Issue5/IJETAE_0513_14.pdf

[2] Cambridge University Computer Laboratory, **The Database of Faces**, http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[3] Sebastian Raschka, http://sebastianraschka.com/Articles/2014_kernel_pca.html#kernel-functions-and-the-kernel-trick

[4] University of Haifa, **Kernel PCA** http://www.cs.haifa.ac.il/~rita/uml_course/lectures/KPCA.pdf

[5] University of Southern Mississippi, **QR Factorization**, http://www.math.usm.edu/lambers/mat610/sum10/lecture9.pdf

[6] Technische Universität Berlin, **Hessemberg matrix visualization**, http://www3.math.tu-berlin.de/Vorlesungen/SS11/NumMath2/Materials/hessenberg_eng.pdf

[7] ETH Zurich, **QR Algoritm**, http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf