DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Reconocimiento de imágenes

21 de Junio de 2013

Métodos Numéricos
Trabajo Práctico 3

Integrante	LU	Correo electrónico
Zajdband, Dan	144/10	dan.zajdband@gmail.com
Ferreria, Manuel	199/10	m.ferreria@gmail.com
Taravilse, Leopoldo	464/08	ltaravilse@gmail.com

Resumen: *Estudio de técnicas de reconocimiento de imágenes mediante transformaciones matriciales usando autovectores.*

Keywords: *Image Recognition, OCR, Eigenvectors, SVD decomposition, covariance matrix applications*

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción teórica

Previo al detalle del experimento realizado se detallan una serie de conceptos teóricos de necesario conocimiento para el entendimiento del trabajo.

1.1. Reconocimiento óptico de caracteres (OCR)

El reconocimiento óptico de caracteres es el proceso de conversión de caracteres de un formato que puede ser escritura a mano alzada, imágenes de libros u otros formatos complejos de entender para una computadora a otro que sea reconocible por ésta.

Este área combina disciplinas como inteligencia artificial, análisis numérico y machine learning. Entre las aplicaciones de OCR se encuentran:

- Digitalización de libros y documentos
- Reconocimiento de tarjetas de crédito y facturas
- Generación de imágenes a partir de documentos digitales
- Resolución automática de CAPTCHA

1.2. Matriz de covarianza

Dado un conjunto de datos de n componentes, se define la matriz de covarianza a la matriz de $n * n$ que contiene en su elemento (i, j) la covarianza entre las componentes i y j de la matriz original.

Siendo la covarianza entre 2 componentes definida como:

$$\Sigma_{ij} = \text{cov}(X_i, X_j) = \text{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right] \text{ con } \mu_i = \text{E}(X_i).$$

1.3. Descomposición en valores singulares (SVD)

La descomposición en valores singulares de una matriz $\mathbf{M} \in \mathbb{R}^{m \times n}$ es una factorización de la forma $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ donde:

- $\mathbf{U} \in \mathbb{R}^{m \times m}$ es una matriz de columnas ortogonales.
- $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ es una matriz diagonal con elementos no negativos y los elementos de la diagonal constituyen los valores singulares de la matriz original.
- $\mathbf{V} \in \mathbb{R}^{n \times n}$ es una matriz ortogonal.

La descomposición en valores singulares tiene diversos usos entre los que se encuentra la resolución de sistemas lineales y la búsqueda de matrices pseudoinversas.

1.4. Introducción al trabajo

El siguiente trabajo consiste en el estudio del mecanismo de reconocimiento automático en imágenes mediante el análisis de componentes principales.

Este tipo de análisis proviene del trabajo llevado acabo en [Sirovich89] y [Turk91]. Estos trabajos reconocen al análisis de componentes principales como herramientas teóricas poderosas a la hora de buscar caracterizaciones automáticas en imágenes. Estos papers se enfocan en el análisis de rostros, una disciplina con un grado de complejidad superior al estudiado en este trabajo.

La técnica se basa fundamentalmente en el hecho que las imágenes en $\mathbb{R}^{n\times m}$ las cuales se quiere reconocer no son variables aleatorias uniformemente distribuidas, sino que existe una función "caja negra" que genera estas imágenes (en nuestro caso, que los dígitos se dibujan de maneras parecidas, como siguiendo un trazo mental).

Como hemos podido observar en el caso de, por ejemplo, la esteganografía, si a una matriz se le alteran las componentes principales de menor magnitúd, la alteración en la matriz reconstruida es baja. Este fenómeno nos da la pauta de que estas componentes aportan menor cantidad de información que las de mayor magnitúd a la hora de representar la imagen.

Una conclusión sacada a priori indica que para reconstruir las imágenes de la forma más fiel posible, y consecuentemente perder la menor cantidad de información posible, es menester conservar la mayor cantidad de componentes principales de la imagen.

El objetivo principal del trabajo es, dada una imágen conteniendo la representación de un dígito, identificar a cual corresponde. La hipótesis asumida a lo largo del trabajo se resume a que las distintas imágenes de un mismo dígito poseen características similares en sus componentes principales. El procedimiento se encarga, dada una imágen cualquiera , de buscar similitudes con los datos de entrenamiento y realizar el matching correspondiente para determinar a que dígito se parece más y así poder identificarlo.

El procedimiento consta, en términos generales, de la generación de una mátriz de covarianza surgente de las imágenes de entrenamiento. Esto se obtiene con una matriz $M \in \mathbb{R}^{n\times m}$, con n cantidad de imágenes de entrenamiento y m cantidad de pixeles por imagen. Luego, se define M como:

$$M_i = \frac{(imagen_i - \mu_{imagenes})}{\sqrt{n-1}}$$

.

Con i una de las filas de la matriz.

Para obtener la matriz de covarianza, se debe calcular M^tM, esto resulta en una matriz $M' \in \mathbb{R}^{m\times m}$. De esta matriz M' es posible extraer sus autovectores, para poder hacer la descomposición apropiadamente. Los autovectores de M' también se pueden obtener como la matriz V de la factorización SVD, $A = U\Sigma V^t$.

Una vez obtenida la matriz V^t de autovectores de la covarianza, se procede a uno de los diferentes métodos de identificación de imágenes.

2. Desarrollo

2.1. Herramientas

Para la realización de los experimentos se utilizaron C++ y Matlab como lenguajes de programación para los algoritmos principales. Durante la realización de este trabajo nos encontramos con la primera barrera en tiempo de corrida de un programa. A la hora de generar la matriz de covarianza esta tarda aproximadamente 5 horas en terminar de definirse debido al gran tamaño de la matriz de entrada.

El análisis de los resultados y el ploteo de los gráficos se realizó mediante el uso del entorno Matlab

Se utilizó LaTeX para la escritura y el formato del informe, manteniendo la estructura de informe del Trabajo Práctico Número 1 y 2 y también permitiendo expresar estructuras y fórmulas matemáticas de una manera simple y clara.

2.2. Manejo de la complejidad del experimento

La dificultad del desarrollo de este experimento fue un poco más elevada a la de los dos trabajos prácticos anteriores.

En primer lugar, se decidió hacer en primera instancia todo el trabajo práctico en Matlab para obtener una pronta aproximación a los resultados, siendo el desarrollo ampliamente más comodo que en C++ (dado que la SVD forma parte de la libreria estandar de Matlab). Una vez implementado el TP en Matlab, se procedió a realizar el experimento en C++. El primer problema encontrado a la hora de implementar el experimento en Matlab fue que al calcular la SVD con 60.000 imágenes, se generaba una matriz de $60{,}000 \times 60{,}000$ elementos de tipo `double`, que no entraba en memoria, es por eso que se decidió utilizar sólo 40.000 imágenes para el experimento en Matlab. Incluso usando como variables no alcanzó a poder realizarse porque Matlab igual la calcula por mas de que enseguida la libere a esa memória.

Luego de desarrollar y contar con el experimento funcionando en Matlab, se empezó la programación del mismo en C++. La primera variación significativa en la implementación se refiere al descubrimiento de que no es necesario calcular la SVD, sino sólo la matriz V, resultando en el cálculo de una matriz de 784×784 en vez de una de $60{,}000 \times 60{,}000$. Aún así, consideraciones posteriores sugirieron que 60.000 imágenes era un número por demás excesivo y por lo tanto el experimento pasó operar con sólo 30.000 imágenes. Al analizar los resultados se corroboró que 30.000 imágenes conforman una muestra significativa, y por lo tanto se realizó el trabajo práctico usando 30.000 imágenes de entrenamiento y 500 imágenes de test.

2.3. Matriz de Covarianza

El primer paso consistió en generar la matriz de covarianza con las 30.000 imágenes tomadas como datos de entrenamiento. Para eso se utilizó la fórmula del enunciado.

Se generó la matriz X a través de la sustracción a cada entrada de la matriz del promedio del pixel que representa (promedio de ese pixel en todas las imágenes) y la posterior división la matriz por $\sqrt{n-1}$ según reza la siguiente formula:

$$(x_i - \mu)^t / \sqrt{n-1} \text{ y, } X = U\Sigma V^t$$

Para calcular la matriz de covarianza, a la que llamamos Mx, se realizó el producto X^tX. Este paso es el acercamiento necesario para realizar el cambio de base computado para la disminución en la redundancia de los datos buscada.

2.4. Factorización QR

A la hora de calcular la SVD se presentó la idea de que sólo es necesaria la matriz V, es por eso que en vez de calcular la SVD, se procedió a calcular la matriz V utilizando la factorización QR de la matriz de covarianza. El método utilizado para calcular la matriz V^t fue la factorización QR de Mx (la matriz de covarianza) mediante la aplicación de transformaciones de Householder, y la posterior multiplicación RQ (es decir, en el orden inverso), hasta que la suma de los elementos debajo de la diagonal principal suman menos de 15. La elección del 15 no fue caprichosa, sino que es el resultado de la corrida del experimento con 10^{-5} en lugar de 15, donde convergía a un número entre 14 y 15 por temas de precisión.

Para calcular la matriz V^t, se utilizó la factorización de Householder y la se implementó en $O(n^3)$ siendo $n = 784$. Como el programa resultante tardó más de 15 horas en calcular la matriz V usando 30.000 muestras, se generó el archivo V.txt con esta información precalculada, si el archivo no existía previamente entonces se genera la matriz y luego se guarda en el archivo, caso contrario, se realiza la lectura del archivo.

Este procedimiento que involucra la generación de una matriz estática tiene sentido en un contexto donde los datos de entrenamiento no se modifican. Otros experimentos donde nuevo input puede ser fuente de mejoras en el método podrían tener problemas pero por las condiciones presentadas no es el caso.

El algoritmo que utilizamos para la factorización QR tiene complejidad $O(n^3)$ ya que para obtener Q hacemos la cuenta

$$Q = I - 2vv^t$$

Como como $R = Q^{-1}A$ y $Q^{-1} = Q^t$ entonces para obtener el producto de las Q calculamos

$$Q_i = Q_{i-1} - (2v)(v^tQ_{i-1})$$

Y este algoritmo resuelve en $O(n^2)$ con esa agrupación en paréntesis y para obtener R hacemos lo mismo con las transpuestas de las Q.

2.5. Cálculo de la Transformación Característica

Para calcular la transformación característica multiplicamos los k autovectores cuyos autovalores asociados eran los de valor más significativo por la imágen de la cual queremos obtener qué dígito representa.

En este punto finaliza la etapa de entrenamiento del sistema. El cambio de base fue realizado acarreando la disminución significativa de información redundante.

2.6. Reconocimiento de dígitos

Para reconocer los dígitos utilizamos dos algoritmos. El primero de ellos fue tomar distancia con norma 2 a todos los dígitos de entrenamiento, y luego comparar con los 100 dígitos más cercanos. El más repetido fue finalmente el elegido para identificar dicha imagen. El otro método que utilizamos fue tomar el promedio de las transformaciones de todos los ceros, de todos los unos, y así con cada dígito, y tomar distancia con norma 2 a cada promedio, quedandonos con el centro de masas que presente la menor distancia.

La idea en el primero es evitar contabilizar las muestras anómalas mientras el segundo hace un análisis del total de la muestra.

3. Resultados

3.1. Iteraciones del cálculo de autovectores

El mecanismo de reconocimiento de imágenes esta basado en los autovectores de la matriz de covarianza, como detallamos antes. Para obtenerlos, se aplica el método iterativo de la factorizacion QR reiterada, descripto anterioremente. La condicion de corte de este método esta basada en la suma de los elementos bajo la diagonal. Estudiamos esta suma y la distribucion de los valores bajo la diagonal, para intentar entender el comportamiento y definir una buena cota.

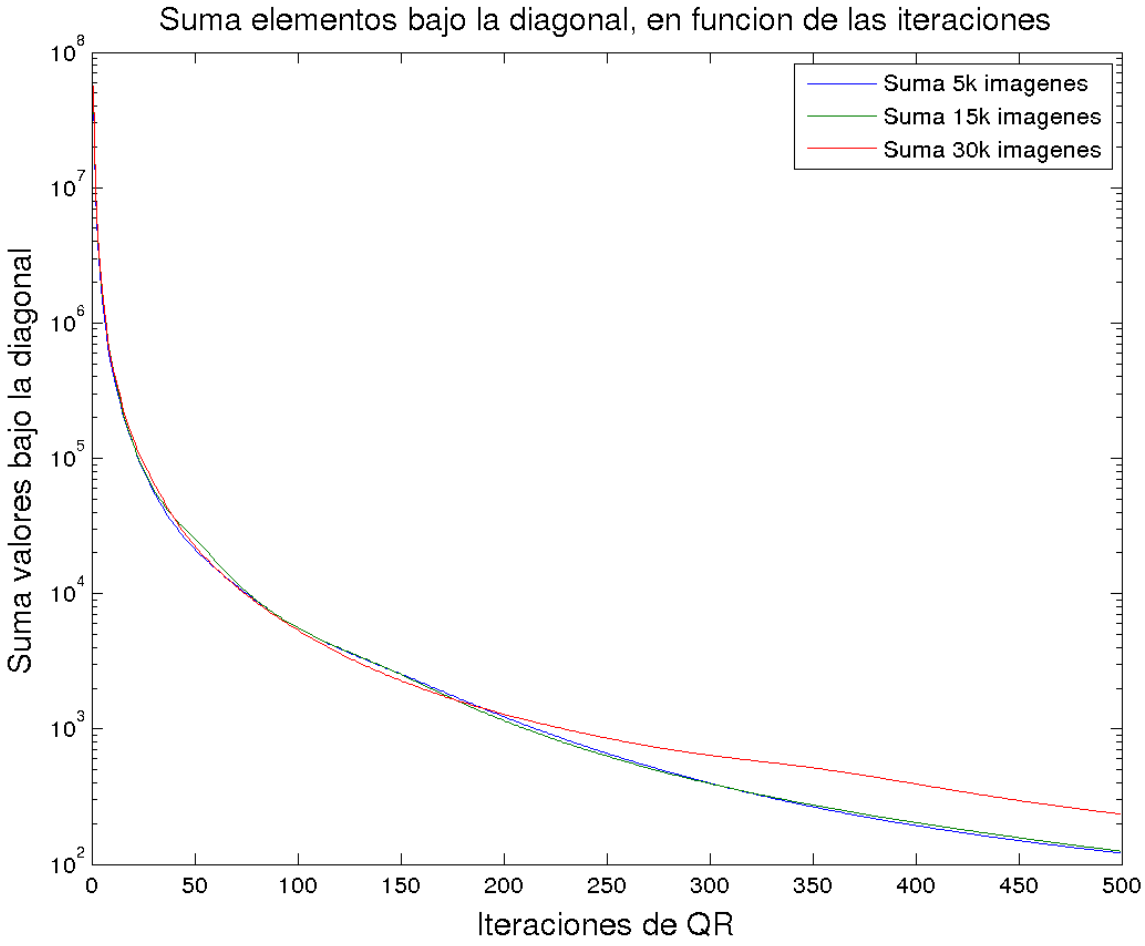

Figura 1: Suma de los elementos bajo la diagonal para 5000, 15000 y 30000 imágenes en funcion de la cantidad de iteraciones de QR.

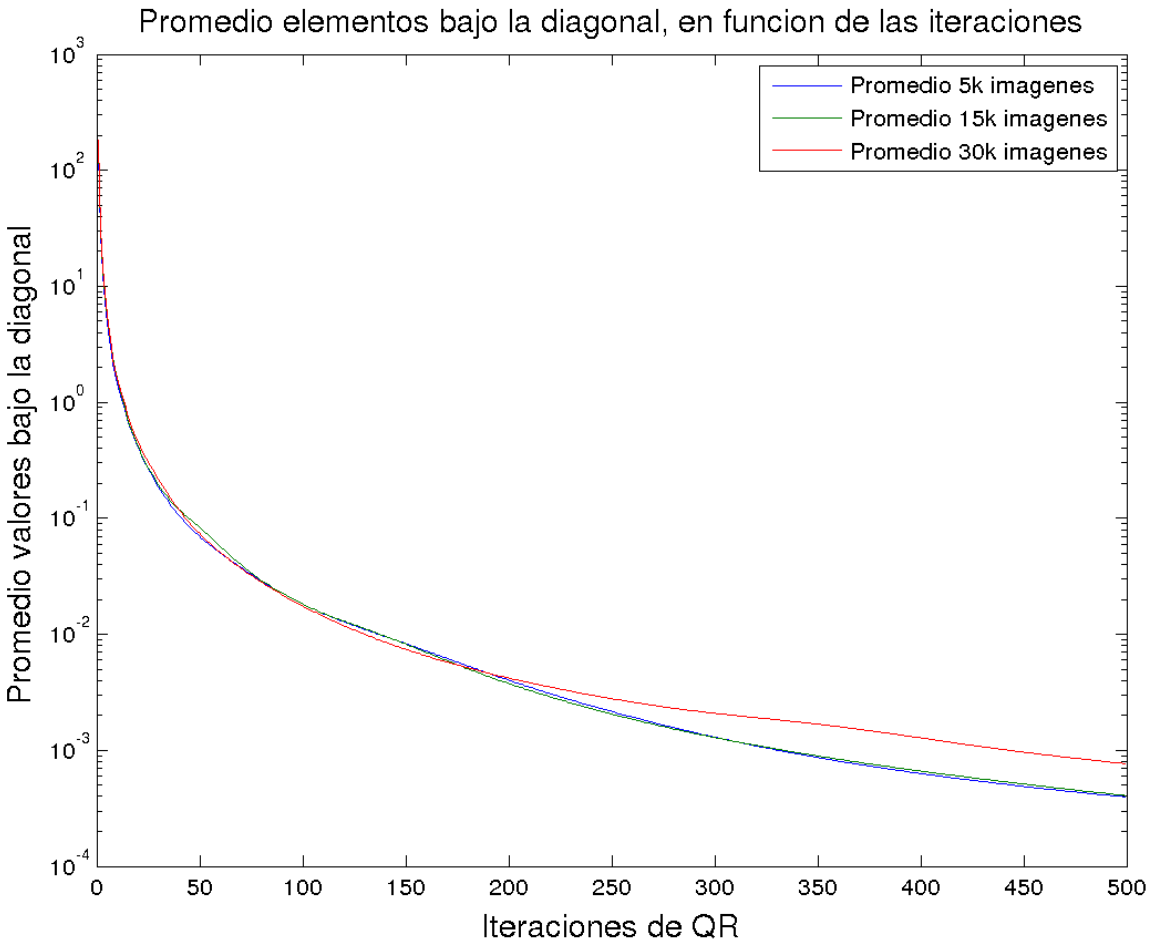

Figura 2: Promedio de los elementos bajo la diagonal para 5000, 15000 y 30000 imágenes en funcion de la cantidad de iteraciones de QR.

3.2. Promedio de Reconocimiento

La métodologia de implementación fue la siguiente: Se construyó una mátriz de covarianza utilizando los primeros n imágenes del dataset `Training Data`. Luego, se tomaban $30000 - n$ imágenes que se descartaban; para poder tomar siempre las mismas t imágenes siguientes como imágenes de test, con sus correspondientes labels. A estos se les aplicaron las técnicas de detección que hemos detallado antes y fuimos variando los k componentes principales de las tuplas que tomabamos para hacer las cuentas de distancia. Los gráficos de HitRate están hechos en función de k. El Hitrate se define como el porcentaje de aciertos en la detección sobre el total de experimentos realizados.

Tomámos siempre las mismas $t = 500$ imágenes para todos los test, que representan de manera proporcional a cada dígito.

Testeamos usando matrices de covarianza entrenadas con cantidad variable de imágenes.

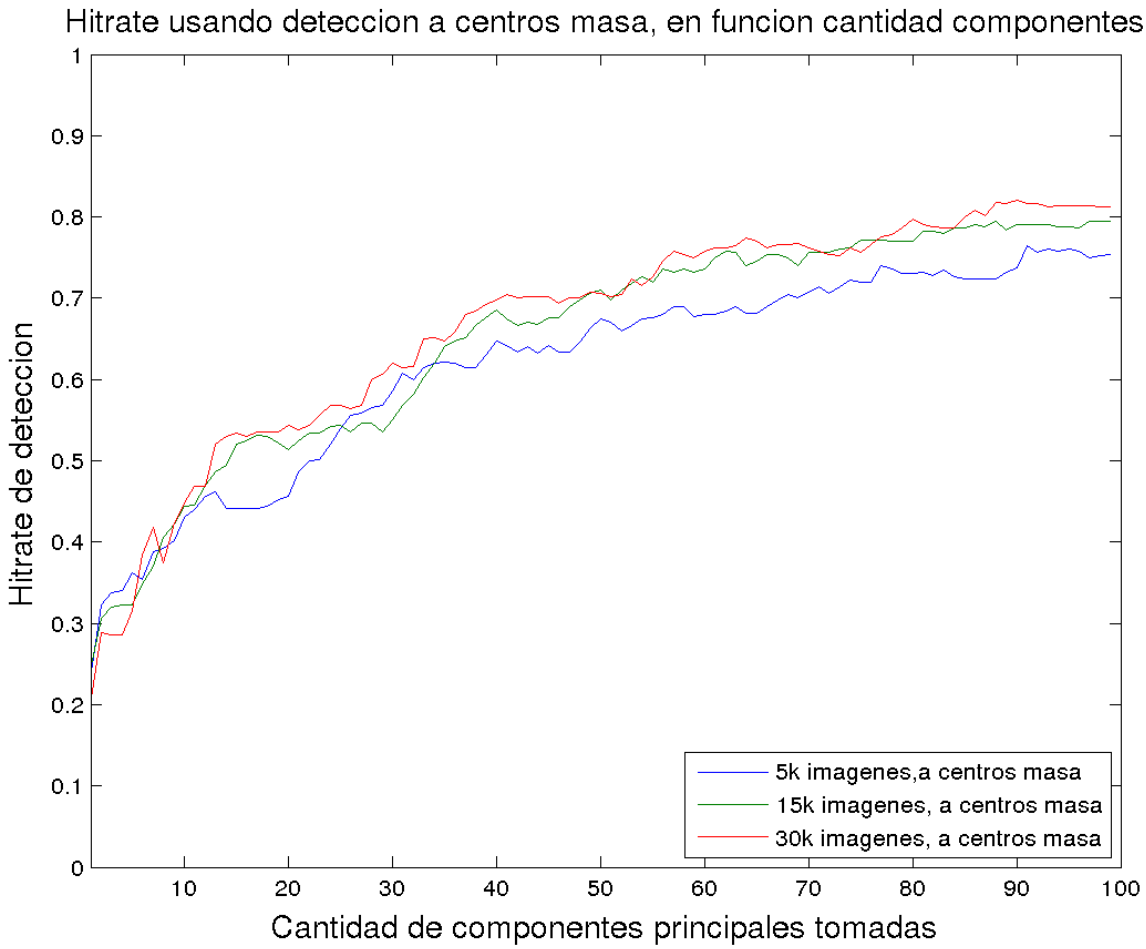

Figura 3: Hitrate de detección de 500 imágenes utilizando la distancia a centros de masa, entrenadas con distinta cantidad de imagénes

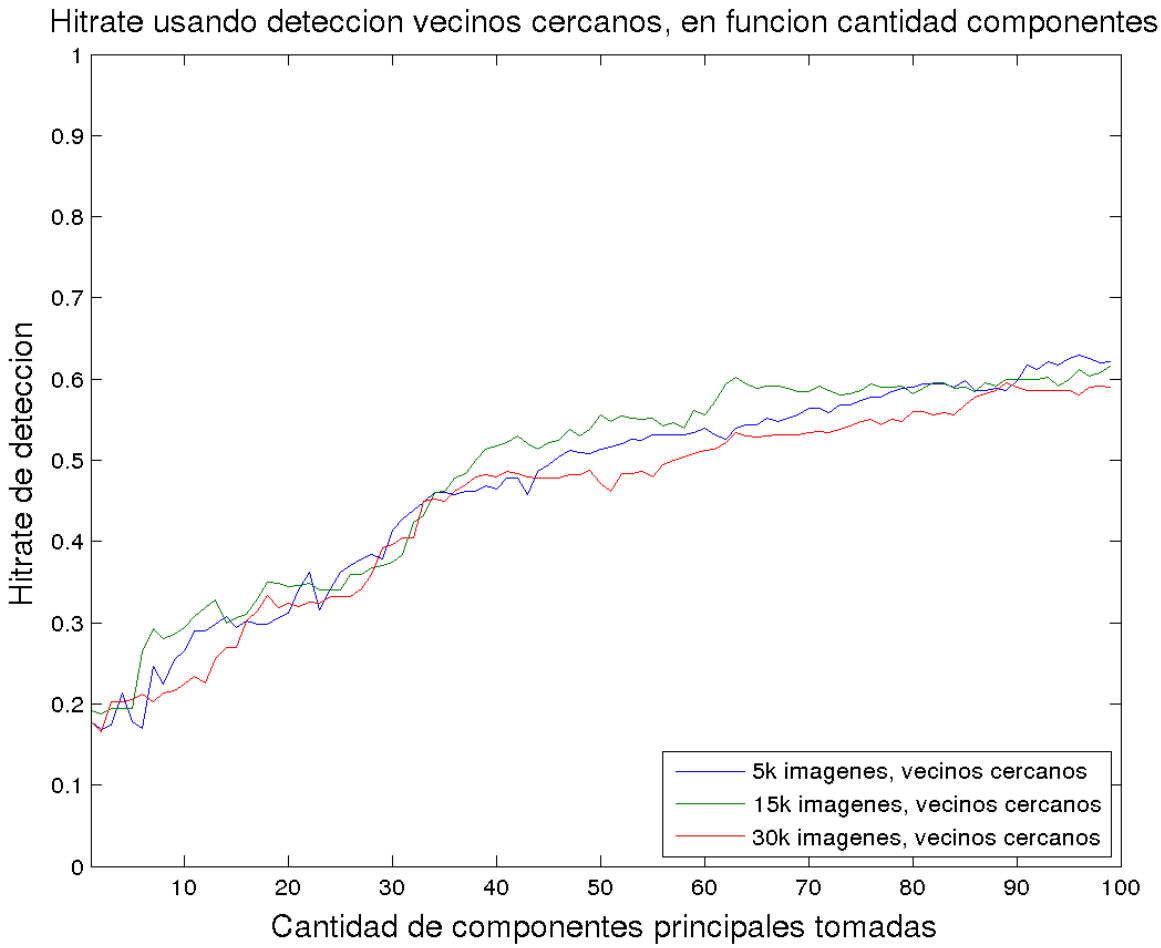

Figura 4: Hitrate de detección de 500 imágenes utilizando método de vecinos más cercanos, entrenadas con distinta cantidad de imaǵenes

3.3. Reconocimiento en funcion de iteraciones

Dados los métodos de detección, nos interesa ver como varia el hitrate de acuerdo a la matriz generada con distinta cantidad de iteraciones de QR. En particular utilizamos el método de distancia a los promedios y 100 vecinos más cercanos, todos medidos usando norma 2.

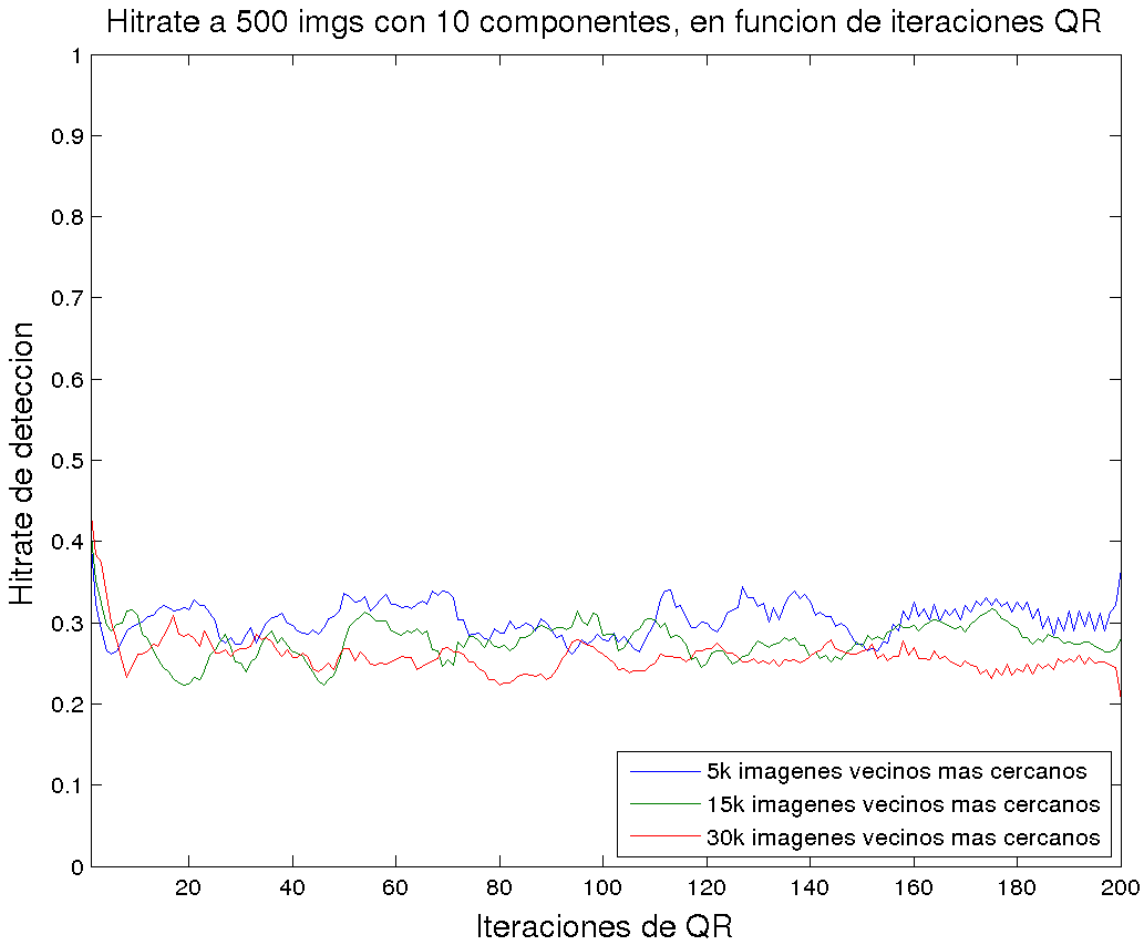

Figura 5: Hitrate en función de la cantidad de iteraciones en la generacion de la matriz usando vecinos más cercanos tomando 10 componentes principales

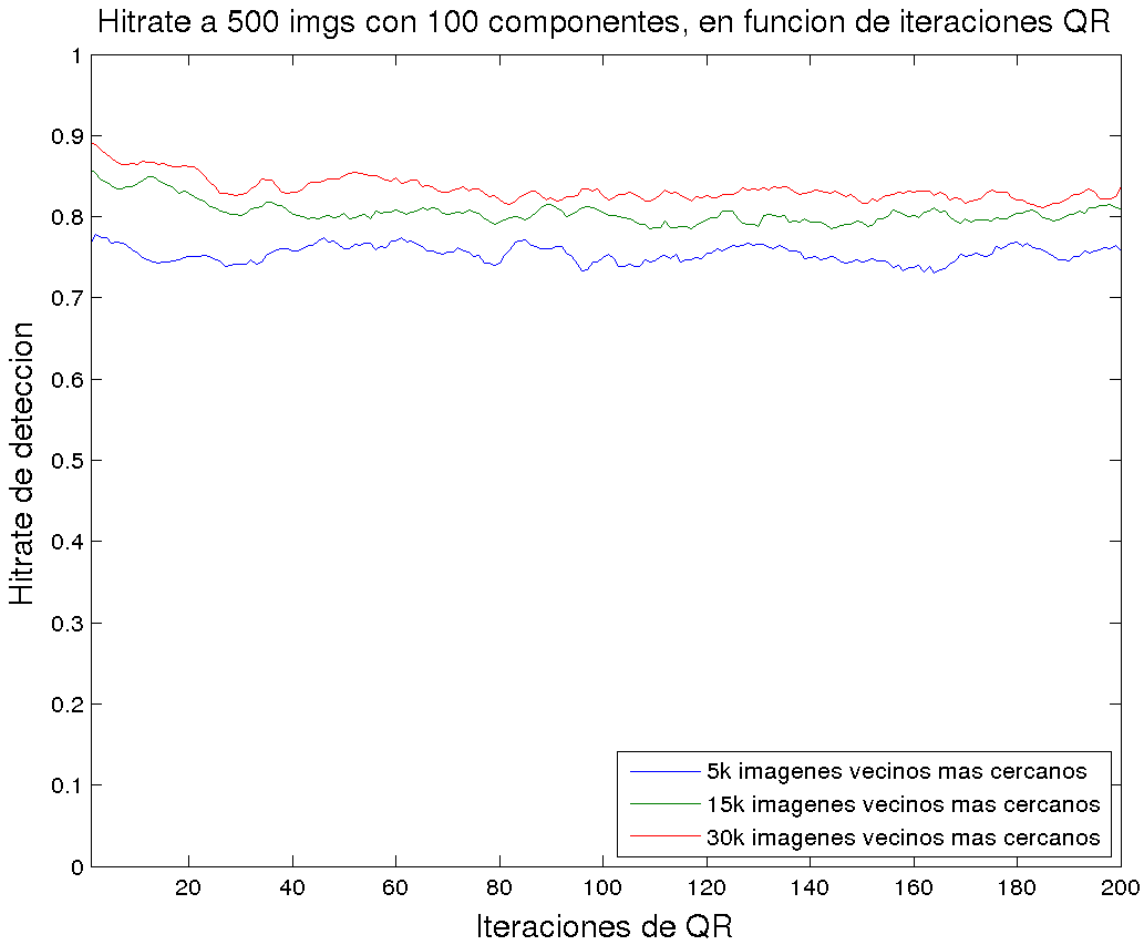

Figura 6: Hitrate en función de la cantidad de iteraciones en la generacion de la matriz usando vecinos más cercanos tomando 100 componentes principales

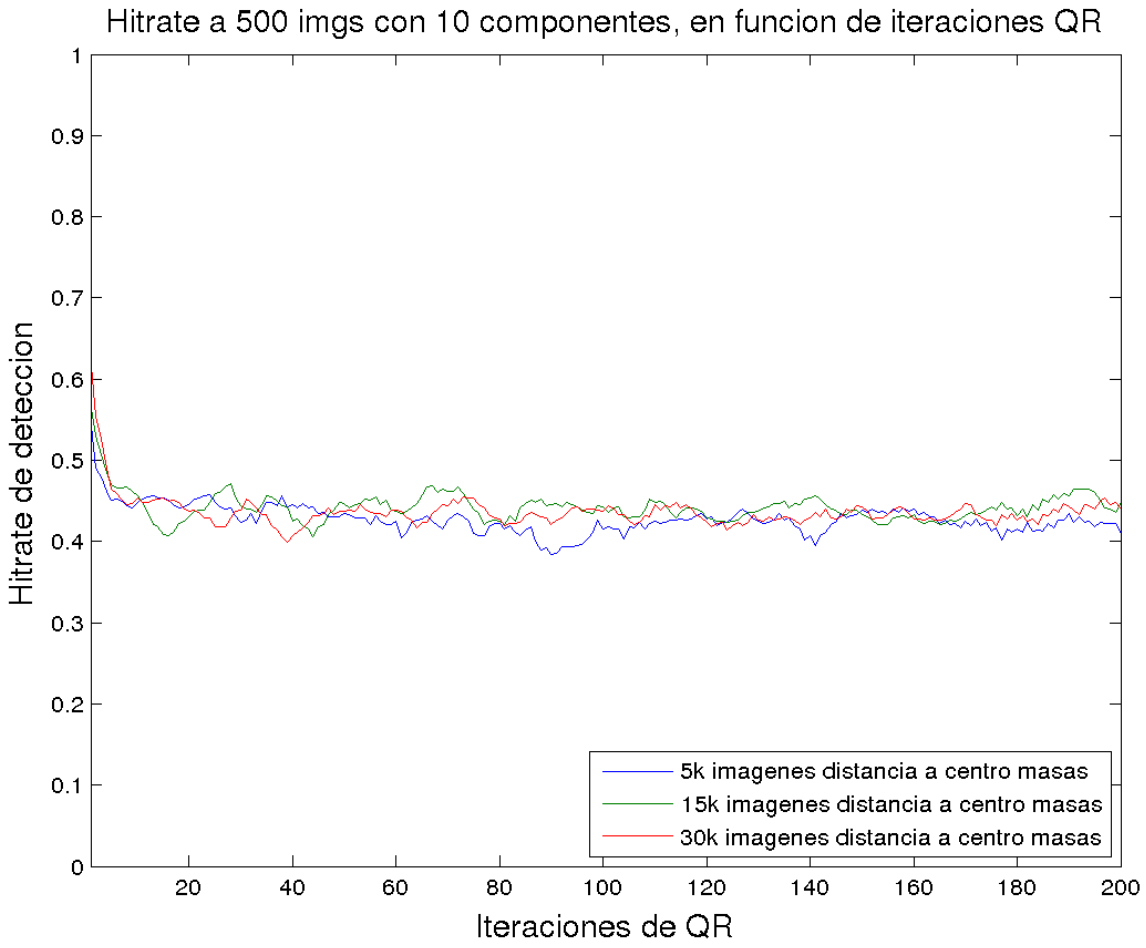

Figura 7: Hitrate en función de la cantidad de iteraciones en la generacion de la matriz usando distancia a los promedios tomando 10 componentes principales

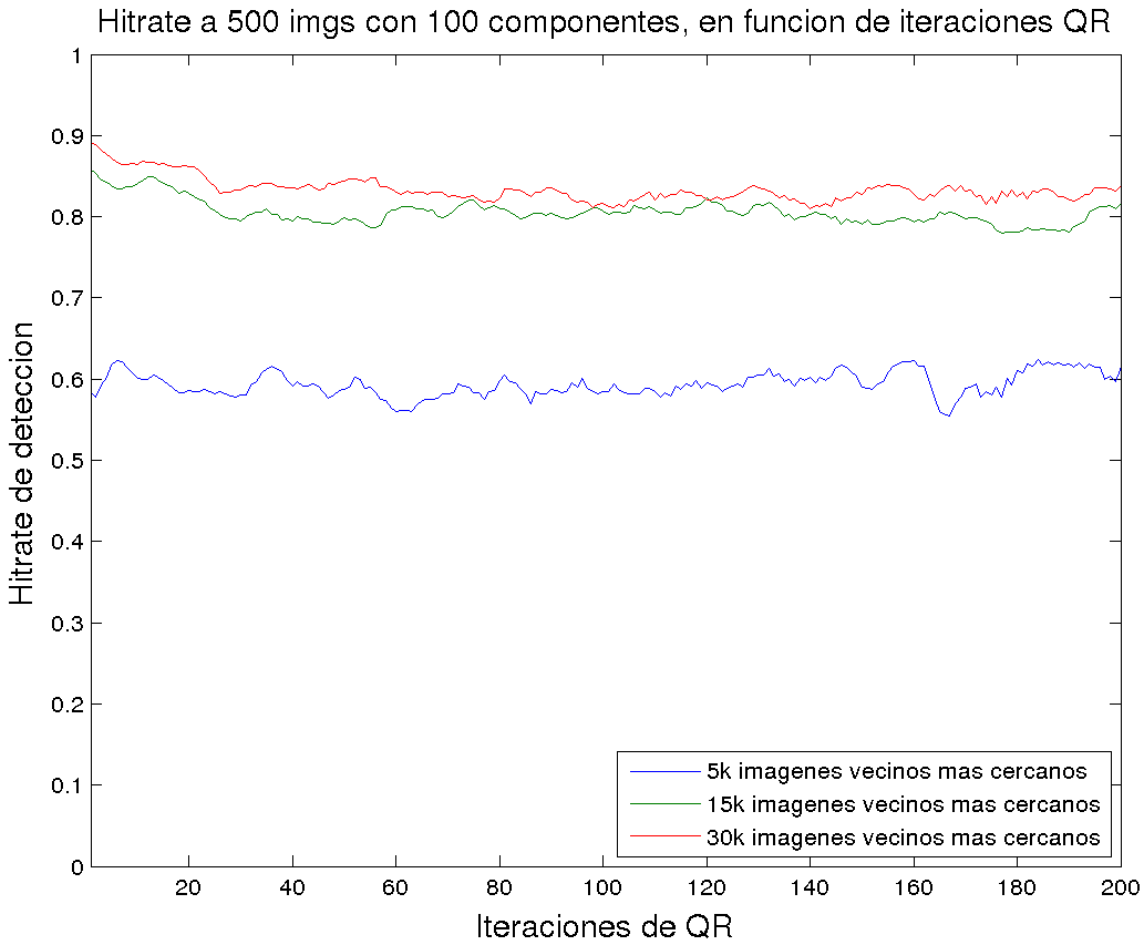

Figura 8: Hitrate en función de la cantidad de iteraciones en la generacion de la matriz usando distancia a los promedios tomando 100 componentes principales

3.4. Reconocimiento por dígito

Fijamos la matriz de covarianza entrenada con 30000 imágenes, ya que está precalculada. Siguiendo la misma metodología, nos concentramos ahora en el HitRate individual de cada dígito. Analizamos el HitRate de acuerdo a la metodología de detección usada.

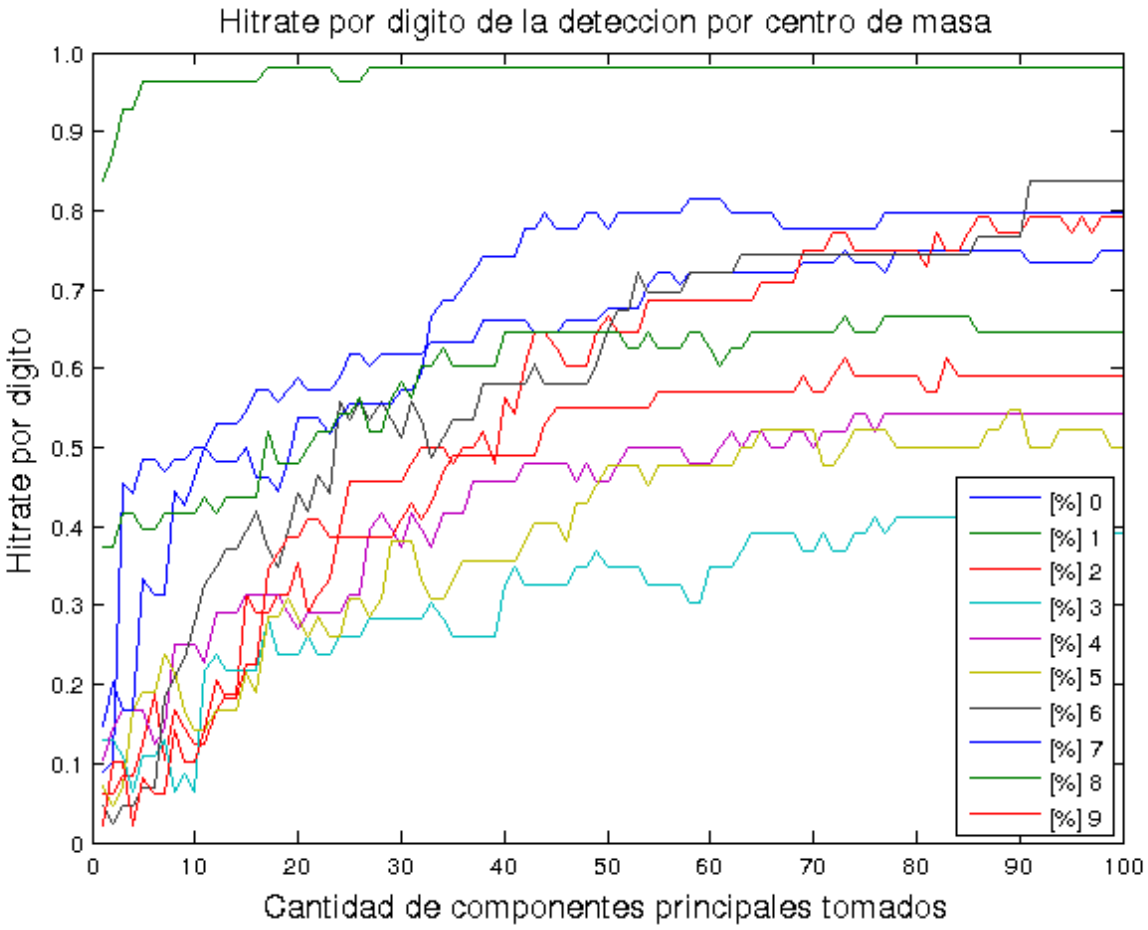

Figura 9: Hitrate por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a los promedios usando distancia a centro de masas

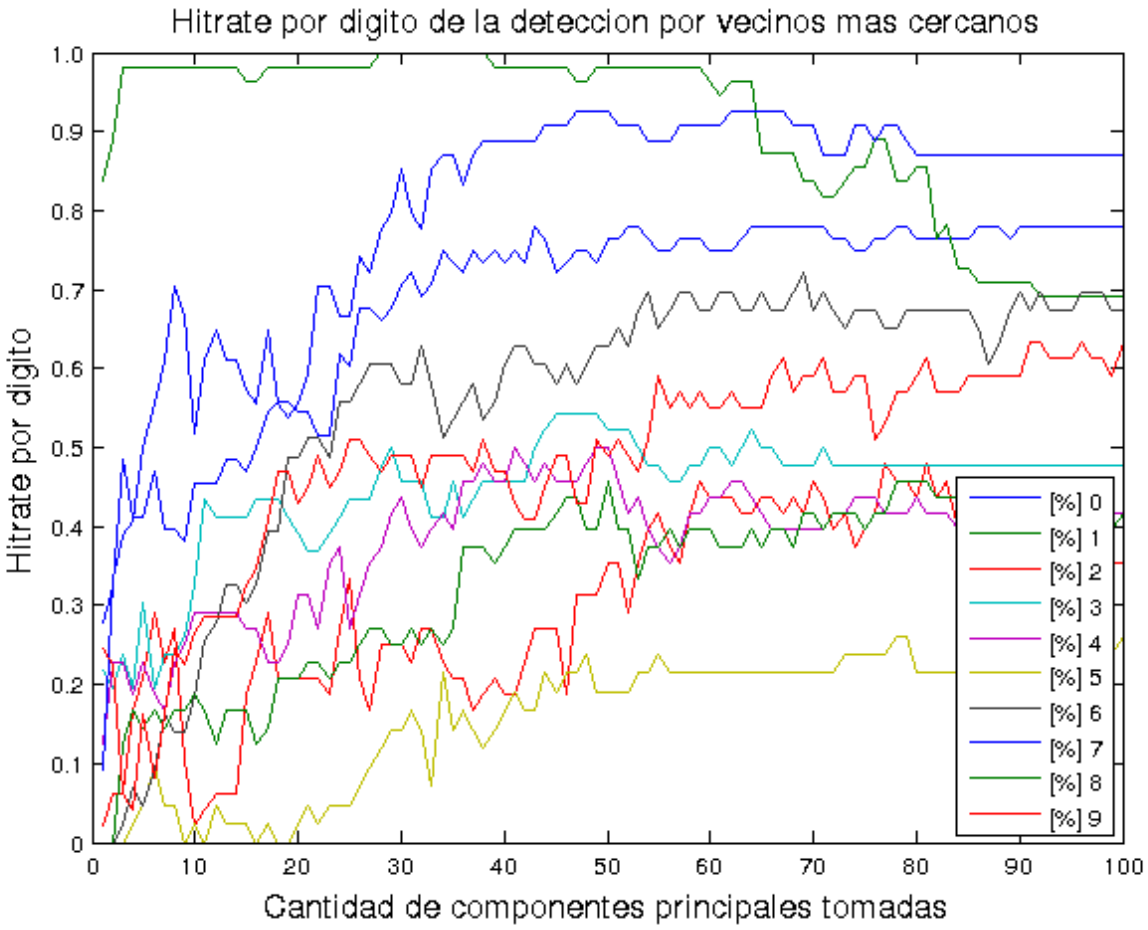

Figura 10: Hitrate por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a vecinos más cercanos

4. Heatmap de reconocimiento por dígito

Aca mostramos las equivocaciones y aciertos al reconocer cada dígito, variando la cantidad de componentes principales tomadas, hecho con la matriz de covarianza de entrenada con 30000 imágenes y variando la cantidad de columnas tomadas. Eje X el valor detectado, eje Y el valor verdadero. El color marca la cantidad de veces que se detectó un valor y a cual correspondia. Los valores negativo son solamente para que se grafiquen mejor, son en realidad positivos.

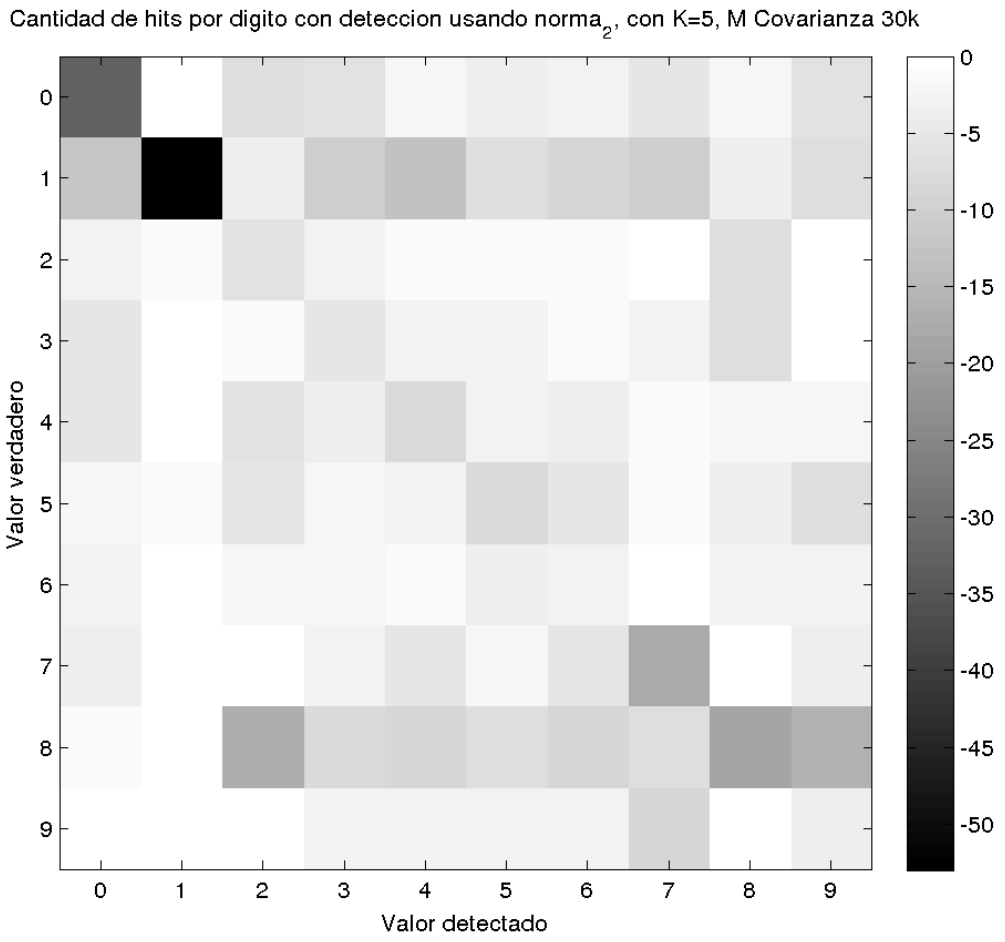

Figura 11: Heatmap de aciertos por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a centros de masas tomando K=5

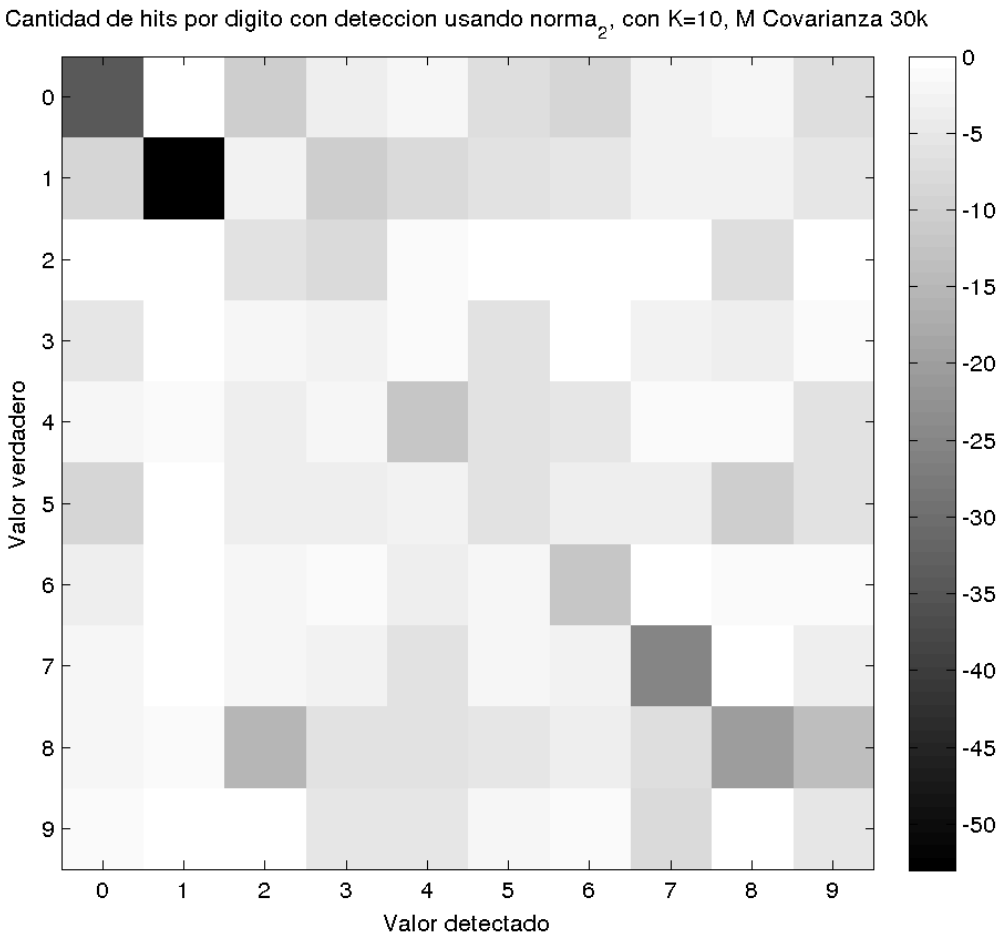

Figura 12: Heatmap de aciertos por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a centros de masas tomando K=10

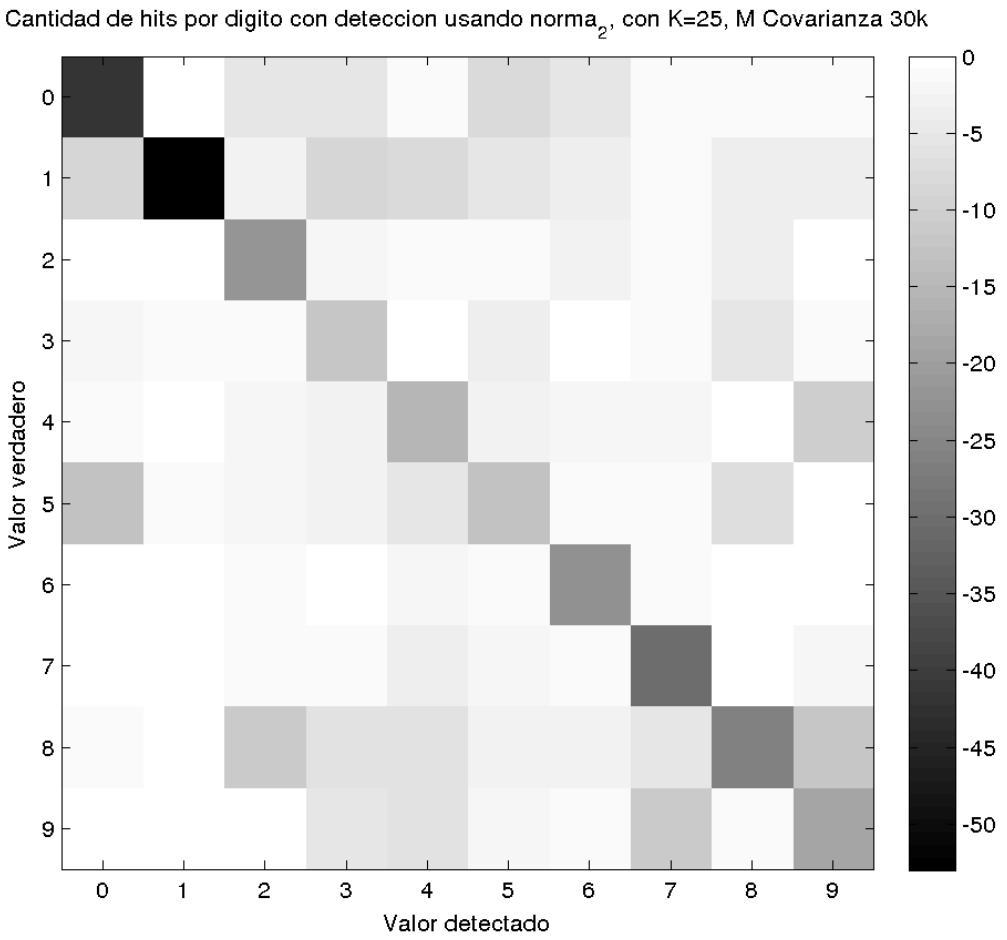

Figura 13: Heatmap de aciertos por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a centros de masas tomando K=25

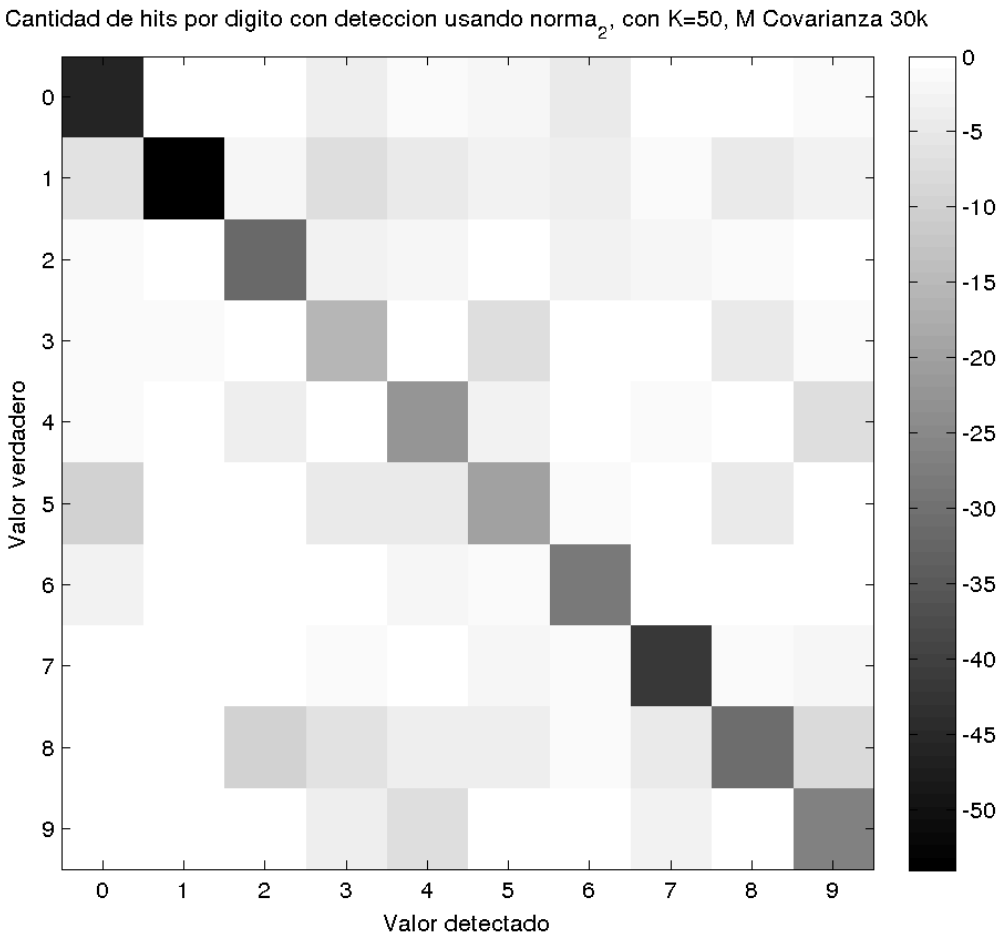

Figura 14: Heatmap de aciertos por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a centros de masas tomando K=50

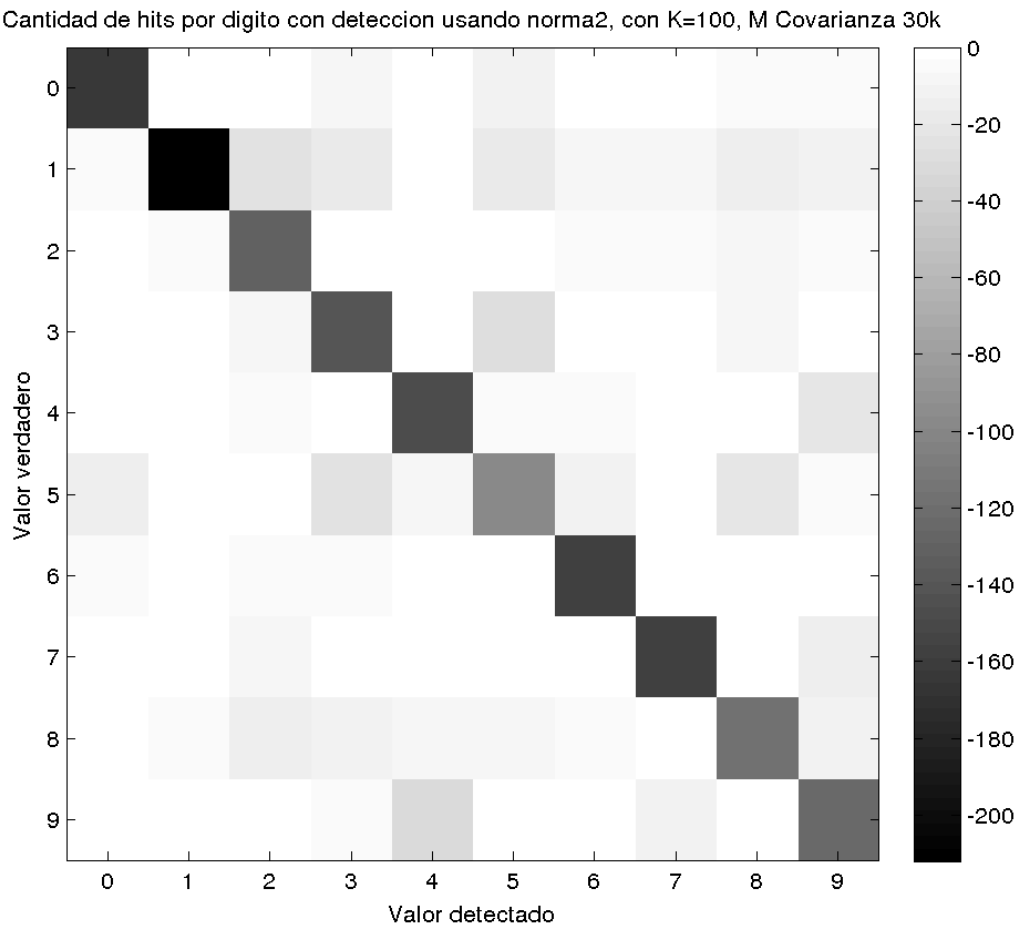

Figura 15: Heatmap de aciertos por dígito de detección de 500 imágenes usando la matriz de covarianza entrenada con 30000 imágenes clasificados por distancia a centros de masas tomando K=100

5. Discusión

5.1. Tamaño de los corpus de entrenamiento

Un punto fundamental del trabajo consistió en el cálculo de autovectores de la matriz de la covarianza de las imágenes. En las secciones anteriores se discutio su construcción y su calculo. Acá comprobamos la relevancia del tamaño del corpus usado a la hora de generarla.

Experimentamos tomando 5000, 15000 y 30000 imágenes (más no usamos porque era realmente excesivo el tiempo de cálculo en nuestro código. Con más tiempo hubiéramos podido paralelizar mejor y poder correr en un cluster o en GPGPU.). Como se puede ver en las figuras 3 y 4, no hay mejores resultados utilizando matrices de transformación basadas en menos imágenes, por lo que si ya hemos generado una grande, conviene seguir usandola mientras tengamos que resolver el mismo problema. Si tenemos que calcularla desde cero, conviene ver cual es el valor que mejora el tradeoff tiempo/performance, ya que las ganancias de detección se vuelven marginales.

El único punto delicado en estos casos, de entrenamiento offline, es tener cuidado de no mezclar datos de entrenamiento con datos de test. Esto podría ocasionar que los hitrates sean excesivamente más altos. Es preferible usar matrices generadas con menos puntos de datos que con más pero "contaminados".

5.2. Iteraciones del cálculo de autovectores

Las matrices de covarianza con las que experimentamos fueron sometidas a analisis de sus valores bajo la diagonal, para ver si efectivamente estaban convergiendo. Pudimos ver que si convergian según los criterios de suma bajo la diagonal, en figura 1. Para definir una cota, los valores de la suma esperabamos que se fueran a cero, pero luego de mas de 500 iteraciones, tendian a un valor cerca de 15. Decidimos comprobar entonces el promedio de valores bajo la diagonal, en la figura 2. Ahi pudimos apreciar que eventualmente casi todos los valores eran cero, salvo muy pocos, que nos estaban forzando a este limite. Decidimos tomar como condicion de corte que la suma de valores bajo la diagonal fuera < 5000, lo cual nos dio alrededor de 100 iteraciones de QR. En efecto estudiamos luego un poco más el hitrate, en función de la cantidad de iteraciones, y pudimos apreciar que 100 iteraciones no variaban significativamente los resultados con respecto a 200 o más.

Un detalle interesante que pudimos observar en las figuras 6 a 8 fue que las sucesivas iteraciones del algoritmo QR de autovectores no presentaba mejorias en la detección. Esto nos sorprendio ya que esperabamos que una matriz que convergiera en sus autovectores, iba a presentar mejor detección que alguna que estuviera menos refinada. Aún mas sorprendete resultó que la mejor detección estaba en las primeras 10 iteraciones, no en las ultimas. Sin embargo, decidimos tomar eso como un fenómeno del corpus y nuestro método, por lo cual para nuestra experimentacio decidimos basarnos en la teoria de detección, que dice que más iteraciones refina mejor.

Nuestra hipotesis es que aplicar cientos de iteraciones conlleva un error númerico grande

(dado que hay $O(n^3)$ operaciones por iteración). Esto hace que se vaya perdiendo precisión numerica y no capte mejor el fenómeno. Lamentablemente no encontramos ninguna referencia a este problema en la literatura, por lo cual no pudimos corroborar esta hipótesis.

5.3. Clasificación de los meťodos empleados

Los métodos que implementamos fueron 2. La norma 2 de distancia a los centros de masas de cada dígito y el más repetido de los 100 vecinos más cercanos al punto.

El método de la norma dos en la figura 3, es sorprendentemente bueno. Comparado con el método de los vecinos más cercanos, es consistentemente mejor para cualquier $k < 100$, que son los más interesantes, ya que idealmente queremos detectar utilizando la menor cantidad de información posbile. Una ventaja de este método, es que solamente hay que comprar contra 10 valores (cada media de dígito), lo cual lo hace enormemente más rapido que el otro método.

Notablemente, la idea de conseguir el más repetido de los 100 vecinos más cercanos en 4 parece tener sentido, pero experimentalmente no es para nada buena para k evaluados, en varios aspectos. No provee una mejor detección, de hecho es consistentemente peor para cualquier k y su runtime además es enorme, ya que debe hacer norma 2 para todos los otros puntos que deseemos considerar, pueden andar en el orden de miles tal vez. Claramente si se desea usar este mecanismo, habría que pensar en mejores estructuras de datos que puedan simplificar el calculo de distancia. Se podria pensar en un algoritmo que haga clustering de datos, que no sea solamente agrupar por dígito, para poder reducir el espacio de busqueda.

5.4. Detección por dígito

Calcular el Hitrate por dígito se nos ocurrió tarde, pero son resultados sumamente interesantes. De la figura 9 a 10 podemos apreciar considerablemente que, dentro de la detección de dígitos, hay algunos que son más dificiles que otros.

Por un lado, el dígito 1 es el más facil de reconocer. Nuestra hipótesis es que no es normal dibujarlo con más de un trazo, y eso simplifica mucho la descomposición en componentes principales, al tener que capturar unicamente ese fenómeno.

Por el otro, el dígito 5 es muchísimo más complicado de reconocer, teniendo hitrates de una fracción de los otros dígitos, incluso con más componentes principales. Esto se debe al fenómeno de que es a veces sutíl la diferencia entre los dígitos como 2 y 8.

En las figuras 11 hasta 14, podemos apreciar en un formato HeatMap como varian los aciertos. Usamos norma 2 para la detección acá ya que antes pudimos apreciar que era método más efectivo. Vemos que cuando tomamos pocas columnas, la diagonal del heatmap (que es el acierto) esta muy difusa, y vemos también otros puntos fuera de la diagonal que tienen un valor bastante alto. Estos son los dígitos que se confunden. Podemos apreciar, por ejemplo, en la figura 11, que los dgitos que más se pueden confundir, con 5 columnas, son el 8 por el 2 y el 4 por el 1.

Seria interesante poder combinar este mecanismo de detección con algun otro basado en un criterio diferente, pero que tenga más poder clasificar entre los dígitos problematicos previamente comentados.

5.5. Cantidad de componentes principales a tomar para la comparación

Un punto clave de este trabajo fue la determinación de la cantidad de valores que debemos utilizar para comparar imágenes. En todas las figuras pudimos observar, en mayor o menor grado, un crecimiento del hitrate con respecto a la cantidad de componentes principales tomados. Obviamente, tomar mas elementos lleva a una mejor caracterización', pero se vuelve un problema de que uno esta básicamente identificando una imagen pixel por pixel. Lo interesante de un sistema de reconocimiento de imágenes es tener una alto grado de detección utilizando la menor cantidad de recursos.

Se puede apreciar tanto en las figuras 11 hasta 15 como en **??**, que hay una diferencia considerable en la elección del método usado, pero sobre todo en la cantidad de componentes principales elegidos. Sin embargo, a partir de ciertos valores ya el crecimiento del hitrate es muy lento ($K = 50$ por ejemplo). Esto en realidad importa poco si la cantidad de imágenes que tenemos que detectar no es grande, la diferencia de cantidad de operaciones que hay entre $K = 50$ y $K = 150$ no es enorme. Dicho esto, hay que tener en cuenta que en sistemas *real-time* esto puede ser excesivamente costoso. Según nuestros experimentos, se deberia hacer un analisis de los requerimientos de forma cuidadosa y ver si el *trade-off* tiempo/precisión es aceptable o no.

6. Conclusiones

Este trabajo nos ha dado la oportunidad de aprender varias lecciones.

6.1. Herramientas de desarrollo ágil

El uso de Matlab ha sido fundamental a la hora de entender el problema. Es muy sencillo perderse en los detalles de la implementación y no ver que problema se quiere resolver. Una herramienta dinámica donde se pueden visualizar en una pasada las factorizaciones de una matriz, las dimensiones y las cuentas de forma matricial sin que importen los detalles del cálculo.

Ayudó también que la velocidad de procesamiento de Matlab es inesperadamente rápida, por lo que se pudo experimentar rápidamente con matrices de covarianza de pocas y de muchas imágenes indistintamente. El costo que se paga por esto es no saber como es la implementación a bajo nivel y no tener control sobre los datos generados. En particular, por ejemplo, si usamos la función `svd` de Matlab, se generan automáticamente las 3 matrices de la factorización, pero solamente nos interesa la V^t (de 784 x 784 elementos). Sin embargo, se genera y guarda tambien la U (de hasta 60000 x 60000 elementos, 26 Gb) y Σ (de hasta 60000 x 784 elementos, 358Mb), llenando innecesariamente la memória RAM, y impidiendonos operar con aún más imágenes usando este mecanismo.

Con las correcciones del TP3, pudimos aprender que si hubieramos calculado la version `'econ'` de `svd`, hubieramos podido los experimentos de casi cualquier tamaño que hubieramos querido. Esto realza aún más la importancia del conocimiento de las herramientas de desarrollo a la hora de enfrentarse a problemas.

La utilización de implementaciones confiables sirvió también para apoyar nuestros cálculos y tener nociones precisas de posibles errores de implementación realizando análisis comparativo de resultados.

6.2. Importancia de los autovectores

Leyendo la literatura de métodos computacionales aplicados al álgebra lineal, es imposible evitar toparse constantemente con los autovectores y autovalores. Su uso atraviesa todas las áreas de los métodos numéricos. Son cruciales en los motores gráficos, a la hora de caracterizar transformaciones típicas. Aparecen en las ciencias físicas como formas de resolver problemas de movimiento, vibración, y fuerzas mecánicas.

El uso dado en este trabajo fue para obtener caraterizaciones de los datos utilizando la descomposición en autovectores, y de esa manera poder correlacionar distintos puntos que tengan diferencias difíciles de describir. Aprendimos que la técnica de descomponer en autovectores es fundamental a la hora de resolver problemas *fuzzy*, donde el análisis de componentes principales puede traer a la luz la verdadera variabilidad de los datos.

6.3. Mecanismo de reconocimiento

Las técnicas que utilizamos para experimentar reconocimiento nos dieron la pauta de que este es el campo que más se puede seguir explorando, sea rastreando la literatura correspondiente como para intentar innovar con algún método único e interesante.

Lo que fue realmente notable es que el uso de ideas que nosotros creíamos más "inteligentes", en realidad resultaron ser peor que la más *naif*, la distancia euclidia a las medias, para casi cualquier cantidad de componentes principales.

Esto nos lleva a pensar que el centro de masas de las imágenes de entrenamiento según lo calculamos nosotros (siguiendo el enunciado y [Sirovich87]) es tal vez una forma óptima de reconocer dada la creación de la matriz de transformación (la matriz V). Si se generase de otra manera esa transformación V^t, creemos que puede haber métodos diferentes que logren un buen reconocimiento también.

6.4. Más puntos de datos, no implican mejores respuestas

La cantidad de imágenes de entrenamiento que debiéramos usar fue un motivo de discusión e hipótesis dentro del grupo. Claramente quisimos usar todas las que pudieramos, intentando emplear al máximo el uso del corpus brindado. La contrapartida fue el tiempo de ejecución. Incluso habiendo implementado la factorización de HouseHolder óptimo ($O(n^3)$), resultaba inaceptable que tardara mas de 24hs el cálculo de los autovectores de la matriz de covarianza con un $\epsilon = 1e - 5$. Implementamos directivas básicas de paralelización que pudieran bajar este runtime (sin cambiar el orden de complejidad del problema). Logramos un speedup de un poco más de 2x con respecto al código serial, usando un procesador con 8 threads, pero el factor cúbico predomina mucho todavía.

Logramos calcular matrices de covarianza para un poco más de 30000 imágenes de entrenamiento, pero cuando corrimos los experimentos, el uso de las matrices generadas con más imágenes superadas las 30000 no aportó casi a un mejor reconocimiento. Viendo esto, decidimos que, al menos con este corpus, no ibamos a lograr captar más detalles del problema, asi que no seguimos intentado crecer en tamaño.

Nuestra hipótesis sostiene que esto es así porque en el corpus está apropiadamente representado cada integrante (dígito). Creemos que en problemas de reconocimiento de rostros, por ejemplo, con muy pocas muestras de cada integrante, sería esencial emplear el corpus al máximo (no necesariamente usarlo todo, pero seleccionar los integrantes inteligentemente) para poder tener todos los detalles posibles para hacer un mejor análisis.

7. Referencias

- Burden, Richard; Faires, Douglas. Análisis numérico. Brooks Cole, 2011.
- Clases de Métodos Numéricos. Departamento de Computación, UBA. Primer Cuatrimestre de 2013.
- Turk, Matthew A., and Alex P. Pentland. "Face recognition using eigenfaces." Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on. IEEE, 1991.
- Sirovich, Lawrence, and Michael Kirby. "Low-dimensional procedure for the characterization of human faces." JOSA A 4.3 (1987): 519-524.

8. Apendices

```
#include<iostream>
#include<algorithm>
#include<vector>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<cassert>

using namespace std;

vector<vector<double> > input, av; //av = autovectores

#define ERRCANT(x) {if(g!=(x)){printf ("Error de lectura %d \n",__LINE__); exit(1);}}
#define ERRCANT(x)

void transpose(vector<vector<double> > &mat)
{
    vector<vector<double>> matrizAuxiliar;
    /** Transposicion de matrices **/
    int n = mat.size();
    int m = mat[0].size();
    matrizAuxiliar.clear();
    matrizAuxiliar.resize(m,vector<double>(n));
#pragma omp parallel for
        for(int i=0;i<m;i++)
    for(int j=0;j<n;j++)
        {
            matrizAuxiliar[i][j] = mat[j][i];
    }
    mat = matrizAuxiliar;
        return;
}

vector<vector<double> > mult(const vector<vector<double> > &A, const vector<vector<double> > &B)
{
    /** Multiplicacion de matrices standard en o(n^3) **/
        int n = A.size();
        int m = B[0].size();
        int t = A[0].size(); // tiene que ser igual a B.size();
    vector<vector<double> > B2 = B;
    transpose(B2);
        vector<vector<double> > res(n,vector<double>(m,0));
#pragma omp parallel for
```

```
        for(int i=0;i<n;i++)
        {
        for(int j=0;j<m;j++)
        {
            double sum =0.0;
                for(int k=0;k<t;k++)
            {
                        sum += A[i][k]*B2[j][k];
            }
            res[i][j]=sum;
        }
    }
        return res;
}

void generateX(vector<vector<double> >& X)
{
        int n = input.size();
        int m = input[0].size();
    vector<double> average(m,0);

    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
                average[j] += input[i][j];
    #pragma omp parallel for
        for(int j=0;j<m;j++)
                average[j] /= (double)n;
    /** Calculo el promedio de cada pixel **/
        X.clear();
        X.resize(n,vector<double>(m));
#pragma omp parallel for
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
                X[i][j] = (input[i][j]-average[j])/sqrt(n-1);
            /** A cada pixel le asigno el pixel en su imagen menos el promedio sobre la raiz de la cantidad de ima
        return;
}

vector<vector<double> > generateMx()
{
    vector<vector<double> > X;
    generateX(X); /** Genero la matriz X **/
    vector<vector<double> > Xt(X);
        transpose(Xt);
        return mult(Xt,X); /** Genero Mx matriz de covarianza como X^t por X **/
```

```
}

double norm(vector<double> &vec) /** Calculo norma 2 de vec **/
{
        double res = 0;
    #pragma omp parallel for reduction (+:res)
        for(int i=0;i<(int)vec.size();i++)
                res += vec[i]*vec[i];
        return sqrt(res);
}

vector<vector<double> > Id(int n) /** Identidad de n x n **/
{
    vector<vector<double> > A(n,vector<double>(n,0));
    for(int i = 0; i< n; i++)
        A[i][i]=1;
    return A;
}



void householder(vector<vector<double> > &A,vector<vector<double> > &Q,vector<vector<double> > &R
{
    /** Factorizacion QR de Householder en o(n^3) **/
        int n = A.size();
    vector<vector<double> > aux(n,vector<double>(n)),aux2(n,vector<double>(n));

        R = A;
        Q = Id(n);
    vector<double> u,v;
    v=vector<double>(n);
        for(int i=0;i<n-1;i++)
        {
        u=vector<double>(i,0.0);
            for(int j=i;j<n;j++)
            u.push_back(R[j][i]);
        double alpha = norm(u);
        if(abs(abs(alpha)-abs(u[i])) < 1e-6) /** Si debajo de la diagonal son todos ceros no itero **/
            continue;
        if(alpha*u[i]>0) /** Cambio el signo si es necesario **/
            alpha *= -1.;
        u[i] += alpha;
        alpha = norm(u);
```

```
#pragma omp parallel for
        for(int j=0;j<n;j++)
            v[j]=u[j]/alpha;

        /** Inicio calculo R **/
#pragma omp parallel for
        for(int j=0;j<n;j++)
            u[j] = 0;

#pragma omp parallel for
        for(int j=0;j<n;j++)
        for(int t=0;t<n;t++)
            u[j] += v[t]*R[t][j];

#pragma omp parallel for
        for(int j=0;j<n;j++)
        for(int t=0;t<n;t++)
            aux[j][t] = 2*v[j]*u[t];

#pragma omp parallel for
        for(int j=0;j<n;j++)
        for(int t=0;t<n;t++)
            R[j][t] -= aux[j][t];
        /** Fin calculo R **/
        /** Inicio calculo Q **/
#pragma omp parallel for
        for(int j=0;j<n;j++)
            u[j] = 0;

#pragma omp parallel for
        for(int j=0;j<n;j++)
        for(int t=0;t<n;t++)
            u[j] += Q[j][t]*v[t];
#pragma omp parallel for
        for(int j=0;j<n;j++)
        for(int t=0;t<n;t++)
            aux2[j][t] = 2*u[j]*v[t];
#pragma omp parallel for
        for(int j=0;j<n;j++)
        for(int t=0;t<n;t++)
            Q[j][t] -= aux2[j][t];
        /** Fin calculo Q **/
        }
        return;
```

```
}

const double delta = 5000;

int iteraciones=0;

bool sigoIterando(vector<vector<double> > &A)
{
        double res = 0.;
        int n = A.size();
    #pragma omp parallel for reduction(+:res)
    for(int i=0;i<n;i++)
    {
        double localRes=0.;
            for(int j=0;j<i;j++)
                    localRes += fabs(A[i][j]);
        res+=localRes;
    }
    printf("Iteracion %d: Suma %lf\t Promedio %lf\n",++iteraciones, res, res/((n*(n-1))/2));
        return res>delta;
        /** Itero hasta que los elementos debajo de la diagonal sumen menos de 15 **/
}
#define MAXITERACIONES 500

vector<vector<double> > Q,R;
vector<vector<double> > allAuVec;
void eig(vector<vector<double> > &A, vector<vector<double> > &auVec)
{
        int n = A.size(); /// A es cuadrada
    vector<vector<double> > matrizAuxiliar;
    auVec = vector<vector<double> >(allAuVec);
    householder(A,Q,R); /** Calculo QR con Householder **/
    A = mult(R,Q); /** Multiplico RQ para obtener la nueva A que es la matriz de covarianza **/
    allAuVec = mult(allAuVec,Q); /** Multiplico todas las Q para obtener los autovectores **/
        /** Ordeno los autovectore segun la magnitud de los autovalores **/
        vector<pair<int,int> > aux(n);

#pragma omp parallel for
    for(int i=0;i<n;i++)
                aux[i] = make_pair(A[i][i],i);
        sort(aux.begin(),aux.end()); /** Ordeno los autovalores **/
        reverse(aux.begin(),aux.end());
        matrizAuxiliar = allAuVec;

#pragma omp parallel for
```

```
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
        {
                auVec[i][j] = matrizAuxiliar[aux[i].second][j];
                /** Asigno a la i-esima columna el autovector correspondiente al i-esimo autovalor **/
        }
        /** Fin ordenamiento de autovectores **/
        return;
}

vector<vector<double> > tc;

vector<double> calctc(vector<double> imagen, int k) /** Calculo la transfomacion caracteristica de imagen co
{
        vector<double> res(k,0);
        int n = imagen.size();

        for(int i=0;i<k;i++)
        for(int j=0;j<n;j++)
        {
                res[i]+=av[i][j]*imagen[j];
        }
        return res;
}

void fillTC(int k) /** LLeno tc con las transformaciones caracteristicas **/
{
        int n = input.size();
        tc.resize(n,vector<double>(k,0));
#pragma omp parallel for
        for(int i=0;i<n;i++)
        {
                tc[i] = calctc(input[i],k);
        }
        return;
}

double dist(vector<double> &v1, vector<double> &v2, int norm)
{
        double res = 0;
        if(norm==0) /** Norma infinito de v1 - v2 | Distancia infinito de v1 a v2 **/
        {
            for(int i=0;i<(int)v1.size();i++)
            res = max(res,v1[i]-v2[i]);
        }
```

```
        if(norm==1) /** Norma 1 de v1 − v2 | Distancias 1 de v1 a v2 **/
        {
            for(int i=0;i<(int)v1.size();i++)
            res += abs(v1[i]−v2[i]);
        }

    if(norm==2) /** Norma 2 de v1 − v2 | Distancia 2 de v1 a v2 **/
    {
        for(int i=0;i<(int)v1.size();i++)
            res += (v1[i]−v2[i])*(v1[i]−v2[i]);
        res = sqrt(res);
    }
        return res;
}

void usage()
{
    cout << "Uso: ./OCR <k> <imp> <norma> <training> <test>" << endl;
    cout << "donde k = cantidad de componentes principales a tomar de las transformaciones." << endl;
    cout << "      imp = 0 (usando nearest neighbours, 1 usando distancia al promedio." << endl;
    cout << "      norma = 0 (norma infinito), 1 (norma 1), 2 (norma 2)." << endl;
    cout << "      training = (default 10000) cantidad de imagenes de entrenamiento, 1 a 30000." << endl;
    cout << "      test = (default 500) cantidad de imagenes de test, 1 a 2000." << endl;
    return;
}

int main(int argc, char* argv[])
{
    if(argc<4){ usage(); exit(1);}
    if(argc>6){ usage(); exit(1);}
    int max_k = atoi(argv[1]), imp = atoi(argv[2]), norm = atoi(argv[3]);
    int min_k = 1;
    /** k es el parametro k del enunciado, imp es la implementacion y norm es la norma que usamos para medir d

    int training_count = 10000; /** Usamos 10000 imagenes de entrenamiento **/
    int test_count = 500; /** Usamos 500 imagenes de test **/
    if(argc>4)
        training_count=atoi(argv[4]); /** a menos que el parametro lo indique **/
    if(argc>5)
        test_count=atoi(argv[5]); /** a menos que el otro parametro lo indique **/
    int padding_count = 30000−training_count; /** paddeamos con imagenes para siempre usar las mismas de tes


        FILE* v = fopen("../datos/trainingImages.txt","r");
        int n, t; /** dims de la matriz de training **/
```

```
int g; /** absorbedora de errores **/
    g = fscanf(v," %d %d",&n,&t);ERRCANT(2);
printf("Se van a leer: %d imgs Training, %d imgs Test \n",training_count,test_count);
vector<vector<double> > testImages;

input.clear();
    input.resize(training_count,vector<double>(t));
    testImages.resize(test_count,vector<double>(t));
/** Comienzo lectura imagenes de entrenamiento y test **/
    for(int i=0;i<training_count;i++)
{
    for(int j=0;j<t;j++)
    {
        g = fscanf(v," %lf",&input[i][j]);ERRCANT(1);
    }
}
for(int i=0;i<padding_count;i++)
{
    for(int j=0;j<t;j++)
    {
        g = fscanf(v," %*lf");
    }
}
for(int i=0;i<test_count;i++)
{
            for(int j=0;j<t;j++)
            {
                    g = fscanf(v," %lf",&testImages[i][j]);ERRCANT(1);
            }
    }
fclose(v);
    /** Fin lectura imagenes de entrenamiento y test **/
    /** Comienzo lectura labels de entrenamiento y test **/
v = fopen("../datos/trainingLabels.txt","r");
g= fscanf(v," %d",&n); ERRCANT(1);
vector<int> labels, testLabels;
labels.resize(training_count);
testLabels.resize(test_count);
for(int i=0;i<training_count;i++)
{
    g = fscanf(v," %d",&labels[i]);ERRCANT(1);
}

for(int i=0;i<padding_count;i++)
{
```

```
        g = fscanf(v," %*d");
    }

    for(int i=0;i<test_count;i++)
    {
                g = fscanf(v," %d",&testLabels[i]);ERRCANT(1);
        }

        fclose(v);
        /** Fin lectura labels de entrenamiento y test **/
        v = fopen("V.txt","r");
        if(v==NULL) /** Si V no existe la genero **/
        {
#ifdef PRECALC
        vector<vector<double> > Mx = generateMx(); // Genero Mx la matriz de covarianza
        allAuVec = Id(Mx.size()); /** Inicializo la matriz de autovectores **/
        bool b = true;
        for(int its=1;its<MAXITERACIONES;its++)
        {

            eig(Mx,av); // Calculo los autovectores de la matriz de covarianza
            b = sigoIterando(Mx);
            if (!b) break;

        }

        v = fopen("V.txt","w"); // Escribo la matriz V en un archivo
        fprintf(v," %d %d\n",(int)av.size(),(int)av[0].size());
        for(int i=0;i<(int)av.size();i++)
        {
            for(int j=0;j<(int)av[0].size();j++)
                fprintf(v," %.6lf ",av[i][j]);
            fprintf(v,"\n");
        }
        fclose(v);
#endif
        }
        else /** Si V ya fue generada previamente la levanto del archivo **/
        {
            int N,M;
            g=fscanf(v," %d %d",&N,&M);ERRCANT(2);
            av.clear();
            av.resize(N,vector<double>(M));
            for(int i=0;i<N;i++)
        {
```

```
            for(int j=0;j<M;j++)
            {
                g=fscanf(v," %lf",&av[i][j]);ERRCANT(1);
            }
        }

        }
#ifndef PRECALC
    vector<vector<double> > Mx = generateMx(); /** Genero Mx la matriz de covarianza **/
    allAuVec = Id(Mx.size()); /** Inicializo la matriz de autovectores **/
    for(int its=1;its<MAXITERACIONES;its++)
    {
        eig(Mx,av); /** Calculo los autovectores de la matriz de covarianza **/
        bool b = sigoIterando(Mx);
#endif
        for(int k=min_k;k<max_k;k++)
        {
            fillTC(k); /** Genero las transformaciones caracteristicas de cada imagen de entrenamiento **/
            vector<double> vec;
            vector<pair<double,int> > distancias;
            vector<int> cant(10);
            int bien = 0;
            int mal = 0;

            if(imp==0)
            {
                distancias.resize(training_count);
                /** La implementacion 0 toma las 100 imagenes mas cercanas y toma la mas repetida de esas 10
                for(int i=0;i<test_count;i++) /** Iteramos sobre la imagenes de test **/
                {
                    vec = calctc(testImages[i],k); /** Calculamos la transformacion caracteristica de la imagen d

                    for(int j=0;j<training_count;j++)
                    {
                        distancias[j] = make_pair(dist(tc[j],vec,norm),j);
                        /** Guardamos en un par la distancia a cada imagen de entrenamiento con la imagen **
                    }
                    sort(distancias.begin(),distancias.end()); /** Ordenamos por distancia **/
                    for(int j=0;j<10;j++)
                        cant[j] = 0;
                    for(int j=0;j<100;j++)
                    {
                        cant[labels[distancias[j].second]]++;
                        /** Contamos cuantas imagenes hay de las 100 mas cercanas con cada label **/
                    }
```

```
            int cualEs = 0;
            for(int j=0;j<10;j++)
                if(cant[j]>cant[cualEs])
                    cualEs = j;
            if(testLabels[i]==cualEs)
                bien++;
            else
                mal++;
        }
    }
    else if(imp == 1)
    {
        /** La implementacion 1 toma el promedio de las transformaciones de cada digito y compara dis
        vector<double> promedio[10];
        int cuantos[10];
        memset(cuantos,0,sizeof(cuantos));
        /** Comienzo calculo promedios **/
        for(int i=0;i<10;i++)
        {
            promedio[i].clear();
            promedio[i].resize(k,0);
        }
        for(int i=0;i<training_count;i++)
        {
            cuantos[labels[i]]++;
            for(int j=0;j<k;j++)
                promedio[labels[i]][j] += tc[i][j];
        }
        for(int i=0;i<10;i++)
        for(int j=0;j<k;j++)
        if(cuantos[i]!=0)
            promedio[i][j] /= (double)cuantos[i];
        /** Fin calculo promedios **/
        for(int i=0;i<test_count;i++)
        {
            vec = calctc(testImages[i],k);
            distancias.resize(10);
            for(int j=0;j<10;j++)
            {
                distancias[j] = make_pair(dist(promedio[j],vec,norm),j);
                /** Comparamos distancia al promedio de cada digito **/
            }
            sort(distancias.begin(),distancias.end());
            int cualEs = distancias[0].second;
            /** Ordenamos y nos quedamos con el mas cercano **/
```

```
                    if(testLabels[i]==cualEs)
                        bien++;
                    else
                        mal++;
                }
            }
            cout << k << " " << bien <<" "<<mal<<" " << (double)bien/(double)(bien+mal)<< endl;
        /** Imprimimos cantidad de hits, cantidad de misses y proporcion de hits **/
        }
#ifndef PRECALC
    }
#endif
    return 0;
}
```

Laboratorio de Métodos Numéricos - Primer Cuatrimestre 2013
Trabajo Práctico Número 3: OCR+SVD

Introducción

El reconocimiento óptico de caracteres (OCR, por sus siglas en inglés) es el proceso por el cual se traducen o convierten imágenes de dígitos o caracteres (sean éstos manuscritos o de alguna tipografía especial) a un formato representable en nuestra computadora (por ejemplo, ASCII). Esta tarea puede ser más sencilla (por ejemplo, cuando tratamos de determinar el texto escrito en una versión escaneada a buena resolución de un libro) o tornarse casi imposible (recetas indescifrables de médicos, algunos parciales manuscritos de alumnos de métodos numéricos, etc).

El objetivo del trabajo práctico es implementar un método de reconocimiento de dígitos manuscritos basado en la descomposición en valores singulares, y analizar empíricamente los parámetros principales del método.

Como instancias de entrenamiento, se tiene un conjunto de n imágenes de dígitos manuscritos en escala de grises del mismo tamaño y resolución (varias imágenes de cada dígito). Cada una de estas imágenes sabemos a qué dígito se corresponde. En este trabajo consideraremos la popular base de datos MNIST, utilizada como referencia en esta área de investigación[1].

Para $i = 1, \ldots, n$, sea $x_i \in \mathbb{R}^m$ la i-ésima imagen de nuestra base de datos almacenada por filas en un vector, y sea $\mu = (x_1 + \ldots + x_n)/n$ el promedio de las imágenes. Definimos $X \in \mathbb{R}^{n \times m}$ como la matriz que contiene en la i-ésima fila al vector $(x_i - \mu)^t/\sqrt{n-1}$, y

$$X = U \Sigma V^t$$

a su descomposición en valores singulares, con $U \in \mathbb{R}^{n \times n}$ y $V \in \mathbb{R}^{m \times m}$ matrices ortogonales, y $\Sigma \in \mathbb{R}^{n \times m}$ la matriz diagonal conteniendo en la posición (i, i) al i-ésimo valor singular σ_i. Siendo v_i la columna i de V, definimos para $i = 1, \ldots, n$ la *transformación característica* del dígito x_i como el vector $\mathbf{tc}(x_i) = (v_1^t\, x_i, v_2^t\, x_i, \ldots, v_k^t\, x_i) \in \mathbb{R}^k$, donde $k \in \{1, \ldots, m\}$ es un parámetro de la implementación. Este proceso corresponde a extraer las k primeras *componentes principales* de cada imagen. La intención es que $\mathbf{tc}(x_i)$ resuma la información más relevante de la imagen, descartando los detalles o las zonas que no aportan rasgos distintivos.

Dada una nueva imagen x de un dígito manuscrito, que no se encuentra en el conjunto inicial de imágenes de entrenamiento, el problema de reconocimiento consiste en determinar a qué dígito corresponde. Para esto, se calcula $\mathbf{tc}(x)$ y se compara con $\mathbf{tc}(x_i)$, para $i = 1, \ldots, n$.

Enunciado

[1] `http://yann.lecun.com/exdb/mnist/`

Se pide implementar un programa que lea desde archivos las imágenes de entrenamiento de distintos dígitos manuscritos y que, utilizando la descomposición en valores singulares, se calcule la transformación característica de acuerdo con la descripción anterior. Para ello se deberá implementar algún método de estimación de autovalores/autovectores. Dada una nueva imagen de un dígito manuscrito, el programa deberá determinar a qué dígito corresponde. El formato de los archivos de entrada y salida queda a elección del grupo. Si no usan un entorno de desarrollo que incluya bibliotecas para la lectura de archivos de imágenes, sugerimos que utilicen imágenes en formato Raw.

Se deberán realizar experimentos para medir la efectividad del reconocimiento, analizando tanto la influencia de la cantidad k de componentes principales seleccionadas como la influencia de la precisión en el cálculo de los autovalores.

Fecha de entrega

- *Formato electrónico:* viernes 21 de junio de 2013, hasta las 23:59 hs., enviando el trabajo (informe+código) a `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP3]` seguido de la lista de apellidos de los integrantes del grupo.
- *Formato físico:* lunes 24 de junio de 2013, de 18 a 20hs (en la clase de la práctica).