Laboratorio de Métodos Numéricos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico Número 2

CSI:DC

Integrante	LU	Correo electrónico
Zar Abad, Ciro Román	129/15	ciromanzar@gmail.com
Lopez Valiente, Patricio	457/15	patricio454@gmail.com
Romero, Lucas Rafael	440/12	lucasrafael.romero@hotmail.com

Reservado para la catedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Resumen

2. Introducción

En este trabajo presentaremos algunas técnicas de clasificación y reconocimiento de patrones. Este tipo de técnicas son muy útiles y usadas en Machine Learning, procesamiento de imágenes, y muchas otras áreas de las ciencias de la computación.

El objetivo principal de este informe es analizar diferentes técnicas de clasificación. En nuestro caso buscaremos clasificar dígitos manuscritos, en particular números, entre 0 y el 9. Para lo cual se dispone de una amplia base de 42000 dígitos ya etiquetados. Cada número manuscrito esta representado por una imagen en escala de grises de 28x28. Por lo que, cada pixel de la imagen contiene un número entre 0 y 255, que indica el nivel de gris de este.

El método de clasificación que emplearemos en este trabajo sera "k-Nearest Neighbours" o mas comúnmente conocido como KNN, el algoritmo KNN, si bien es eficaz en la clasificación de elementos con características similares a los nuestros, el tiempo de computo que requiere esta directamente relacionado con la dimensión de estos, por lo se presenta como una característica deseada que la dimensión de nuestros elementos sea lo suficientemente pequeña para que el tiempo de cómputo también lo sea y al mismo tiempo lo suficientemente grande para que KNN funcione de manera efectiva. Para lograr estos objetivos complementaremos KNN con herramientas de pre procesamiento, que realizaran la tarea de reducir la dimensión de los elementos, al mismo tiempo capturando la información imprescindible de estos, como "Principal Component Analysis" y "Partial least squares Discriminant Analysis", comúnmente conocidas como PCA y PLS-DA respectivamente.

3. Desarrollo

3.1. Métodos utilizados

3.1.1. KNN

Este algoritmo abstrae cada elemento como un punto en espacio. Este también se relaciona a una clase, en nuestro caso un numero. Por lo que se logra representar todos los elementos de nuestra base de datos, cada uno relacionado a su clase correspondiente. De esta forma, cuando se tiene que clasificar un nuevo elemento, se procede a buscar los k vecinos más cercanos y se lo asigna a la clase que represente a la moda de estos, es decir la clase con mayor cantidad de representantes en los k mas cercanos.

3.1.2. PCA / PLS-DA

Los objetivos de PCA y PLS-DA son similares. Estos algoritmos transforman el espacio en el que los elementos se encuentran, transformándolos a un espacio donde

la información importante de cada elemento se encuentre en las primeras variables. Esto permite reducir la dimensión, y al mismo tiempo minimizar la perdida de información. Para esto lo que buscan nuestros métodos es capturar aquellas características que contribuyen mas a su varianza. Por lo que redefinen nuestro espacio original en uno nuevo de dimensión reducida que contiene la mayor cantidad de información posible. La diferencia principal entre estos dos métodos es que mientras PCA no tiene en cuenta la clase a la que pertenecen nuestros elementos, PLS-DA si lo hace para mejorar su transformación.

3.2. Implementación

La implementación de los métodos utilizados se realizo en C++, la realización de la misma si bien no tuvo grandes inconvenientes, fue importante en el proceso para comprender que realizar algoritmos que optimicen las operaciones en función de la dimensión de los elementos a procesar, es de vital importancia para reducir los tiempos de computo.

4. Experimentos

Al poseer una base de 42000 dígitos correctamente etiquetados, decidimos usar para los siguientes experimentos, excepto se explicite en el mismo, técnicas de çross validation", en particular el "K-fold cross validation", ya que estas dan resultados estadísticamente más robustos. para ello designamos dos tipos de particiones "K-foldçon K=5 y K=10. Las razones por las que elegimos estas particiones son:

1. K = 5, elegimos esta partición arbitrariamente como control ya que nos parecía que tener como validación el 80 % de los datos era una buena medida.

2. K = 10, a partir de la necesidad experimental de ver cuanto afecta a una medida el K elegido, por lo que seleccionamos este K porque no solo era lo suficientemente distante a 5, sino que también preservaba un buen caudal de datos de test.

Cabe aclarar que en los casos que corresponda, esto es que presenten "K-fold cross validation" en sus bases experimentales, las mediciones de tiempos de cómputo, excepto se indique lo contrario, son efectuadas sobre la resolución de todos los folds, por lo que si se quisiera obtener el tiempo de cómputo real correspondiente a una base como la de nuestros experimentos, asumiendo que la distribución de los tiempos de computo para cada fold es uniforme, bastaría con dividirlo por el numero de folds. Para la medición de los mismos se emplearon la librerías "*chrono*" y "*ctime*" de $c++$, estas mismas cuentan con diversas funciones para la medición de tiempos dentro del código, fueron empleados estos procesos y no otros conocidos como el comando "*time*" de bash u otros, ya que al permitir mediciones en el código no solo ofrecían a priori un resultado mas fiable y menos permeable a procesos externos, sino que dada su maleabilidad facilitaban enormemente la toma de mediciones.

4.1. Metodo de la potencia

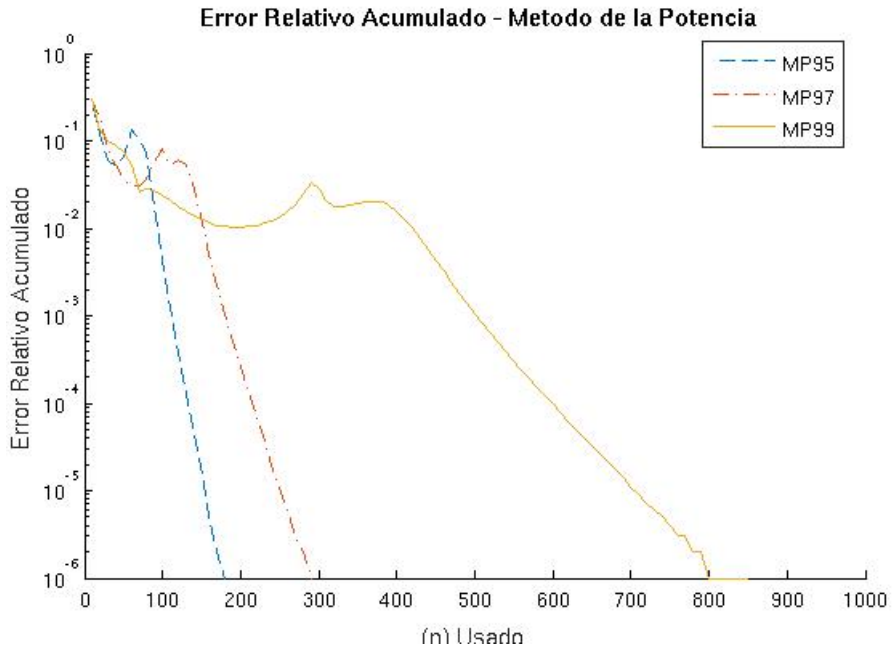
4.1.1. Desarrollo

Este experimento pese a estar poco relacionado con el reconocimiento de patrones a priori, se presentaba de vital importancia para dar confianza a los resultados obtenidos a los experimentos posteriores a este, ya que era conocido que iba a ser necesario el calculo de autovectores y que estos eran de imprescindibles en nuestros métodos de pre procesamiento, por lo que como primer experimento quisimos verificar el nivel de exactitud y por ende, de error de este método.
Para ello era conocido dada la información provista por la cátedra al inicio de este trabajo que el orden de convergencia de este método esta dado por el cociente:
$\theta = \mid \frac{\lambda_2}{\lambda_1} \mid$
Siendo λ_1 el mayor autovalor de la matriz y λ_2 el subsiguiente, de forma tal que mientras mas cercano sea θ a 1 , mayor sera la cantidad de iteraciones necesarias para obtener de manera precisa el autovector asociado a λ_1.
Para trabajar en un ambiente controlado se diseñaron tres matrices MP99, MP97 y MP95, donde la matriz MP$\alpha \in \mathbb{R}^{100x100}$ es una matriz diagonal, por lo que los autovalores están en la diagonal y esta definida de la siguiente forma:
$mp_{ii} \in$ MPα,
$mp_{11} = 100$,
$\forall\ 2 \leq \text{i} \leq 100\ mp_{ii} = mp_{i-1i-1} \times \alpha$
De esta forma se obtiene que el orden de convergencia $\theta = \alpha$ para cada autovalor buscado. Se eligió una matriz de 100x100, ya que si bien nuestras matrices son de una dimensión notablemente mayor la cantidad de autovectores a encontrar difícilmente superan los 100 y con los tiempos de cómputos obtenidos se puede realizar una extrapolación para obtener una medida razonablemente confiable.
Para analizar la exactitud del método se calculo los 100 autovectores y posteriormente sus correspondientes autovalores de las matrices previamente nombradas y se calculo el error relativo acumulado de los autovalores con respecto a los reales, esto es la suma del modulo de los errores relativos de los 100 autovalores, esto realizo para parámetros $n = 10$ a $n = 1000$ del método de la potencia. Ademas se midió el tiempo para calcular todos los autovalores para cada n.

4.1.2. Resultados

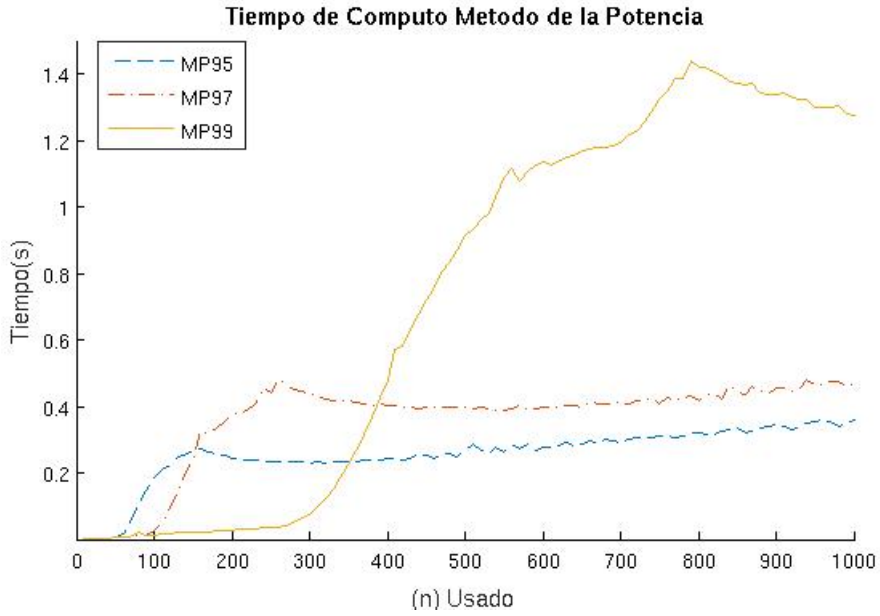
En la Figura 1 se puede observar el orden del error del método, mientras que en la Figura 2 se observan el tiempo de computo de este.

Figura 1. Error Relativo Acumulado



En la Figura 1 se observa como se comprueba la hipótesis de que θ influye notablemente en el orden de convergencia y los beneficios de usar un n "grande", también se puede observar que a partir de $n = 800$ el error pareciera inexistente aunque esto se debe al orden de precisión de nuestras mediciones.

Figura 2. Tiempo de Computo



En la Figura 2 si bien se observan algunas medidas que parecen anormales, ya que para mayor cantidad de iteraciones requiere menor tiempo de computo, atribuimos esto a un outlier por sobrecarga del sistema, ya que los demás medidas concuerdan

con esta teoría. Ademas se observa que θ tiene incidencia en el tiempo de computo algo que no era esperado en un principio, ya que se pensaba que solo dependía de n.

A partir de las experiencias realizadas, considerando que 5 decimales de precisión es algo deseable y que los tiempos de computo eran razonables se eligió como 700 como n para los siguientes experimentos.

4.2. Knn

4.2.1. Desarrollo

Este experimento se llevó a cabo para evaluar los tiempos de cómputo, y la efectividad del método knn, sin haberle aplicado antes a las imágenes ninguna transformación para reducir las dimensiones.

Experimentamos sobre distintos valores de la variable k (parámetro de knn) y la cantidad de Kfolds en la que dividimos la base de datos.

Los valores de k los variamos entre el minimo posible, 1, y 50. Nos detuvimos en 50 ya que notamos que los resultados seguían una tendencia evidente para valores mayores.

Los tiempos están expresados en segundos, y para medir la efectividad del método utilizamos como medidas el Hitrate, el menor de los Recall de cada dígito, el promedio de los diez Recall, el menor de los Presition de cada dígito, y el promedio de los diez Presition. En cada uno de estos, se expresa el promedio de los resultados de cada Kfold.

4.2.2. Resultados

En la Figura 3 y la Figura 4 se pueden observar los distintos resultados de la efectividad del método.

Luego, en la Figura 5 se observan el tiempo que le llevó a la misma computadora llegar a estos resultados, sin tener en cuenta otras etapas del proceso como por ejemplo cargar las imágenes en memoria.

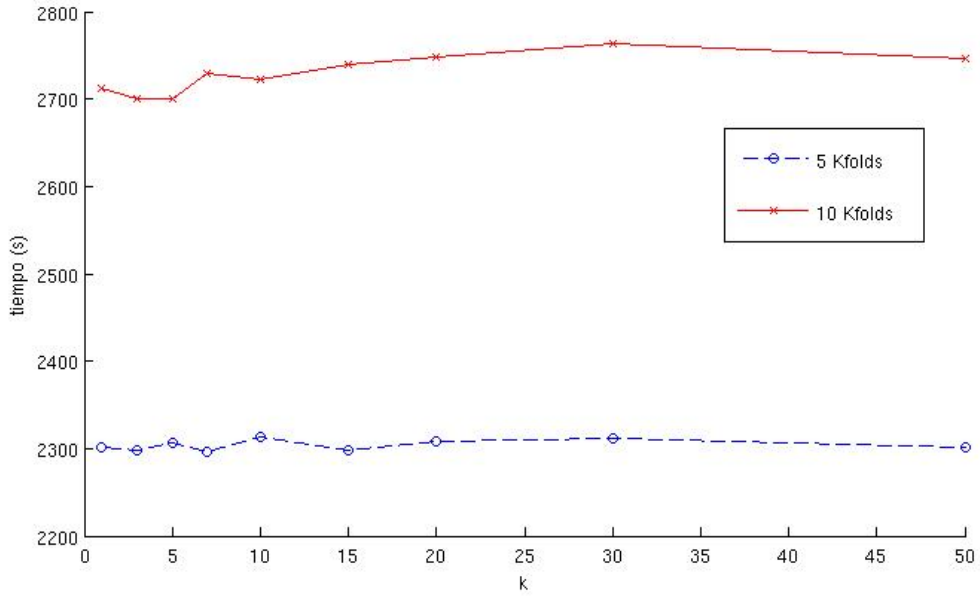
Figura 3. Efectividad de knn con 5 Kfolds

K	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.966833	0.930271	0.966402	0.937964	0.967011
3	0.954262	0.902974	0.953833	0.847080	0.956369
5	0.934357	0.848532	0.933710	0.776954	0.939984
7	0.915429	0.801251	0.914544	0.719537	0.925327
10	0.890000	0.728373	0.888792	0.652254	0.907431
15	0.852167	0.636189	0.850333	0.572861	0.882310
20	0.818952	0.553152	0.816641	0.515260	0.862062
30	0.762405	0.427837	0.759311	0.436992	0.830001
50	0.676143	0.269929	0.671940	0.353074	0.781640

Figura 4. Efectividad de knn con 10 Kfolds

K	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.967690	0.928948	0.967182	0.939080	0.967853
3	0.955238	0.904664	0.954825	0.854410	0.957104
5	0.936738	0.852516	0.936121	0.784250	0.941908
7	0.919143	0.807044	0.918332	0.729261	0.928212
10	0.894357	0.739987	0.893162	0.663791	0.910228
15	0.857738	0.643996	0.856069	0.582617	0.885904
20	0.826048	0.571033	0.823889	0.525791	0.866585
30	0.771262	0.448028	0.768350	0.446857	0.835709
50	0.689095	0.289420	0.685123	0.362779	0.789873

Figura 5. Tiempos de Knn



4.2.3. Discusión de los resultados

De estos resultados se puede observar claramente que, en el caso de que la base de datos no haya pasado por ninguna transformación antes, k=1 es el valor mas efectivo.

Además, dividiendo la base de datos en 10 Kfolds los resultados mejoran levemente. Esto nos sorprendió, ya que pensamos que como la base de datos estaba desordenada, la cantidad de Kfolds no debería haber influenciado positiva o negativamente en la efectividad del experimento.

Por último, como se puede ver en la Figura 5, el parámetro k que se elija no parece influenciar significativamente en los tiempos, que se comportan de forma constante, aunque si se ve afectado por la cantidad de Kfolds.

A partir de estos resultados, decidimos que para los próximos experimentos vamos a utilizar los tres valores de k que nos dieron mejores resultados, 1, 3 y 5. El

motivo por el que no nos quedamos solamente con k=1, es que si bien en el contexto de la base de datos de entrenamiento original fue el óptimo, en los métodos PCA y PLS se realizan transformaciones que podrían hacer que esto cambie. Elegimos tres valores y no más, porque preferimos enfocarnos en experimentar sobre otras variables más concernientes a cada uno de los posteriores experimentos.

4.3. PCA

4.3.1. Desarrollo

En este experimento evaluaremos la efectividad y la velocidad de este método en distintas circunstancias. Para eso experimentaremos entre los valores de k que mejor resultado nos dieron en experimentos anteriores, dos Kfolds, y valores de α entre el menor posible,1, hasta 50, que consideramos que debería ser suficiente, y para α's mayores los tiempos eran muy elevados.

Éste método utiliza entre sus algoritmos el método de la potencia. En un experimento previo se llegó a la conclusión de que un criterio de parada de 700 ciclos tiene una precisión y aceptable en un tiempo de cómputo no demasiado elevado, por lo que, al igual que en los demás experimentos que lo precisen, nos atendremos a este valor sin variarlo. Para esto estamos suponiendo que el contexto de este experimento puntual, no difiere mucho del utilizado en el experimento del método de la potencia.

4.3.2. Resultados

En principio, experimentamos con distintos valores de α, utilizando los tres mejores k que obtuvimos de un experimento previo. En la Figura 6 se puede observar la comparación de los Hitrates obtenidos.

Luego, en la Figura 7 y Figura 8 se puede ver el detalle de la efectividad, con k=1, que fue el que mejor resultados nos dio nuevamente.

Para sacar una conclusión final, comparo los mejores resultados de cada Kfold, en ambos casos k=1 y α=50, en la Figura 9.

En cuanto al tiempo que tarda ejecutar este método, se expresan en la Figura 10 los tiempos que tardó una misma computadora en llegar a los resultados antes expresados. A los tiempos expresados en el gráfico, hay que sumarle 1761.62 y 4092.65 segundos para las corridas de 5 y 10 Kfolds respectivamente, que es lo que llevó calcular las 5 y 10 matrices M correspondientes. Para agilizar los experimentos éstas se calcularon previa e independientemente de las mediciones de tiempo.

Figura 6. Hitrate de PCA con 5 Kfolds

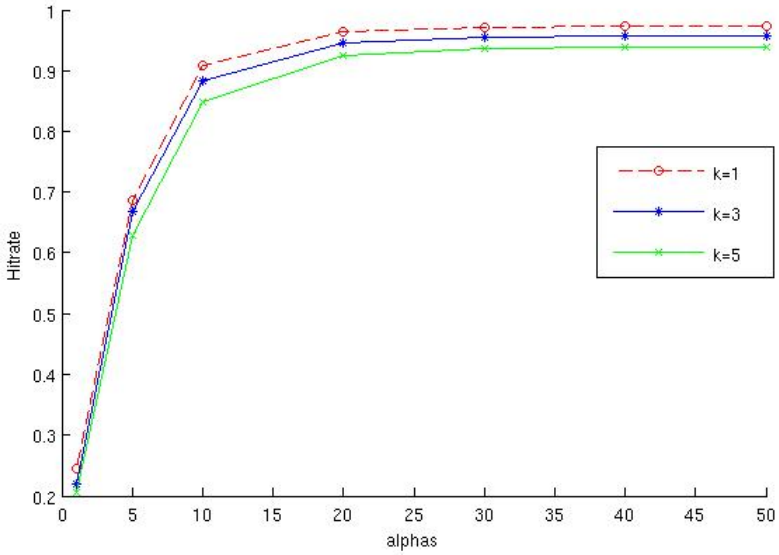


Figura 7. Efectividad de PCA con 5 Kfolds, k=1

Alpha	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.245286	0.106383	0.238065	0.104752	0.238625
5	0.687714	0.531089	0.683826	0.534819	0.683515
10	0.909524	0.823155	0.908432	0.814193	0.908387
20	0.963238	0.932885	0.962941	0.921434	0.962974
30	0.971810	0.950213	0.971604	0.943140	0.971639
40	0.972286	0.951589	0.972058	0.947021	0.972110
50	0.972381	0.953049	0.972177	0.944309	0.972293

Figura 8. Efectividad de PCA con 10 Kfolds, k=1

Alpha	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.242238	0.095239	0.234933	0.096988	0.234296
5	0.689024	0.535112	0.685154	0.535777	0.684856
10	0.911262	0.826707	0.910191	0.810956	0.910258
20	0.964857	0.930573	0.964560	0.924620	0.964532
30	0.972857	0.945441	0.972625	0.944068	0.972711
40	0.973000	0.948053	0.972745	0.946130	0.972921
50	0.973357	0.948814	0.973122	0.946571	0.973260

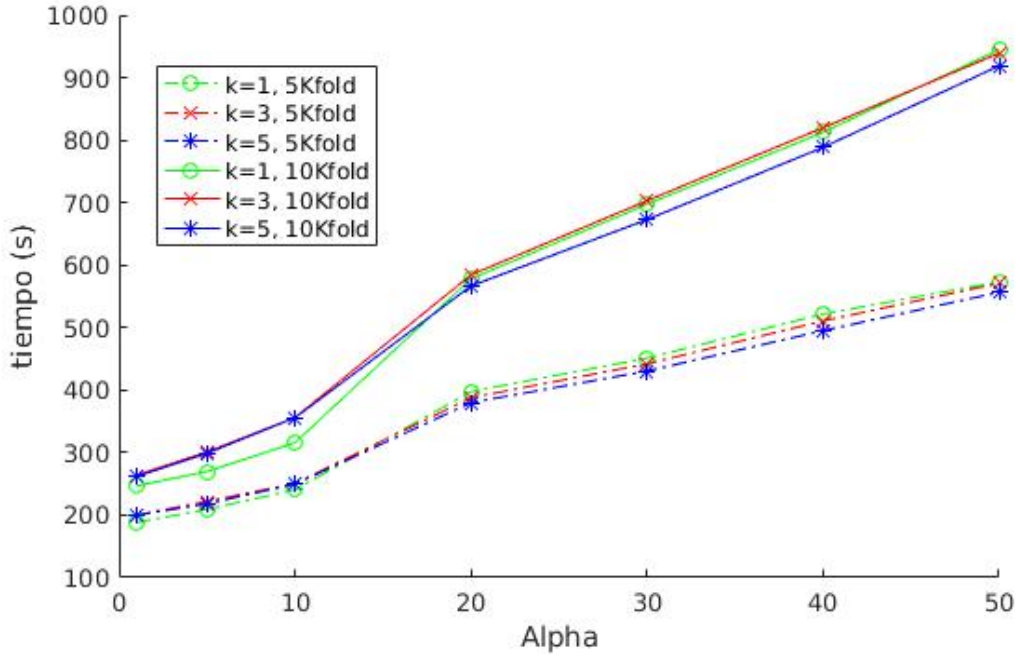
4.3.3. Discusión de los resultados

En este experimento, se sigue observando que la mejor efectividad se da al comparar las imágenes solo con un único vecino más cercano en knn.

Figura 9. Efectividad de PCA con k=1 y alpha=50

Kfolds	Hitrate	Menor Recall	Recall Promedio	Menor Presition	Presition Promedio
5	0.972381	0.953049	0.972177	0.944309	0.972293
10	0.973357	0.948814	0.973122	0.946571	0.973260

Figura 10. Tiempos de PCA



También se mantiene la tendencia que se vio marcó en el experimento de knn, en el cual observábamos que al testear con 10 Kfolds los resultados daban un mejor hitrate. Sin embargo, los Recall son mas disparejos, ya que a pesar de que el promedio es ligeramente mejor, el menor Recall es significativamente mas bajo.

Esto se puede deber a que estamos testeando con una partición de solo el 10 % de la base de datos, y en algunos casos para algún dígito puede tratarse de una muestra poco significativa.

Entre los valores de alpha elegidos, el más alto, 50, fue el que dio mejores resultados, aunque a partir de 40 la diferencia parece ser casi insignificante. En este experimento no quedó determinado cual es el valor de alpha óptimo, ya que las estadísticas de efectividad podrían seguir subiendo. Ésto nos motivó a realizar un experimento posterior, en el que evaluaremos como se comporta el método para valores de alpha mayores a 50.

4.4. PLS-DA

4.4.1. Desarrollo

En este Experimento evaluaremos tanto la efectividad de PLS-DA como técnica de pre procesamiento de nuestras imágenes y su respectivo impacto en el posterior reconocimiento de las mismas. También evaluaremos el tiempo de computo de este método.
Así mismo como este método usa fuertemente el "método de la potencia". asumiremos como parámetro de este 700 por las razones vistas en el experimento correspondiente y en la posterior predicción del dígito usa "Knn", como queríamos ver como afectaba el k elegido a la precisión del método y si esta variación se correspondía con la del espacio original, asumiremos como "k" 1, 3 y 5, ya que estos demostraron ser los mas efectivos previamente, y aunque en este caso estemos trabajando sobre un espacio transformado, por lo que no es trivial que los mas efectivos sigan siendo estos, como decisión experimental decidimos que aportaba mayor riqueza trabajar sobre el parámetro de PLS-DA, osea γ. Asimismo también haremos uso de "K-folds", siendo estos los que declaramos previamente.
Para ello experimentaremos sobre una amplia gama de parámetros γ, observando para cada uno de ellos tanto el tiempo de computo, como métricas conocidas como: Hitrate, Recall y Precisión. Donde consideramos:

1. Hitrate
2. Menor Recall, el minimo Recall de cada dígito.
3. Recall Promedio, el promedio de los diez Recall.
4. Menor Precisión, el minimo Precisión de cada dígito.
5. Precisión Promedio, el promedio de los diez Precisión.

En cada uno de ellos, se expresa el promedio de los resultados de cada K-fold.

4.4.2. Resultados

Tanto en la Figura 11, como en la Figura 12 se puede observar que a medida que aumenta nuestro γ también lo hace nuestra efectividad en todo sentido, aunque parece estacionarse o estar cerca un punto critico, esto es un máximo, asimismo también se puede observar que el impacto que tienen folds elegidos en esta es despreciable. Por ultimo también cabe destacar que si bien no lo hacen en la exacta relación anterior el impacto que tiene "k" sobre la efectividad es prácticamente el mismo que presentaba en el espacio original.

Figura 11. Efectividad de PLS-DA con 5 Kfolds

K = 1 / 5 Folds					
Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.259238	0.117798	0.253116	0.121815	0.252892
5	0.765	0.599659	0.761874	0.60868	0.761033
10	0.92169	0.863092	0.920564	0.862399	0.920694
20	0.961714	0.932255	0.961376	0.91821	0.961428
30	0.967667	0.94493	0.967387	0.936697	0.96744
40	0.970595	0.948924	0.970384	0.93966	0.970427
50	0.97169	0.951043	0.971496	0.943044	0.971552
K = 3 / 5 Folds					
Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.23319	0.011533	0.22952	0.076338	0.286288
5	0.742405	0.439863	0.740208	0.51306	0.773215
10	0.8965	0.761692	0.895403	0.742291	0.903242
20	0.942738	0.865921	0.942425	0.818298	0.945892
30	0.953452	0.89346	0.953176	0.846816	0.955507
40	0.955595	0.901427	0.955325	0.854314	0.957405
50	0.957548	0.905249	0.957306	0.861375	0.959156
K = 5 / 5 Folds					
Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.215071	0.002393	0.212344	0.03166	0.277104
5	0.703595	0.307167	0.700941	0.457044	0.760533
10	0.86769	0.690061	0.866395	0.674674	0.883339
20	0.919905	0.80643	0.919419	0.748579	0.928063
30	0.9335	0.847545	0.933102	0.77846	0.939211
40	0.936429	0.851056	0.936007	0.78426	0.941833
50	0.938524	0.857598	0.938193	0.795422	0.943335

Figura 12. Efectividad de PLS-DA con 10 Kfolds

K = 1 / 10 Folds					
Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.259619	0.111044	0.253519	0.108345	0.254218
5	0.766048	0.601404	0.762863	0.608328	0.762161
10	0.923738	0.859979	0.92264	0.868266	0.922618
20	0.961905	0.927471	0.961528	0.920393	0.961618
30	0.968429	0.937501	0.968115	0.933045	0.968215
40	0.972214	0.944891	0.971963	0.940409	0.972075
50	0.972976	0.947851	0.972705	0.944675	0.972897
K = 3 / 10 Folds					
Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.232	0.010063	0.228335	0.077545	0.28163
5	0.744619	0.449671	0.742379	0.515403	0.774249
10	0.899833	0.764534	0.898846	0.74759	0.906358
20	0.943976	0.866287	0.943673	0.821279	0.946986
30	0.954167	0.894699	0.953881	0.846535	0.956221
40	0.957095	0.901146	0.95683	0.858554	0.958732
50	0.959595	0.906718	0.959374	0.868388	0.961017
K = 5 / 10 Folds					
Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.214476	0.002855	0.211755	0.034131	0.273994
5	0.704952	0.308217	0.702342	0.45959	0.760492
10	0.869	0.692428	0.867749	0.676592	0.884355
20	0.922095	0.808773	0.921615	0.754158	0.929695
30	0.935595	0.846239	0.935169	0.784227	0.940979
40	0.938024	0.850544	0.937607	0.792081	0.942945
50	0.941238	0.863463	0.940821	0.802126	0.945534

En laFigura 13 y la Figura 14 observamos que si bien los folds no tenían mayor impacto en la efectividad del método, si lo tienen y en gran medida sobre el tiempo de cómputo del mismo, ademas se puede observar claramente que el costo de transformar nuestro espacio no es bajo, y es inversamente proporcional a nuestro γ, sin embargo uno de los beneficios que presenta, es que aliviana enormemente el calculo de Knn, como se esperaba.

Figura 13. Tiempo de Computo de PLS-DA con 5 Kfolds

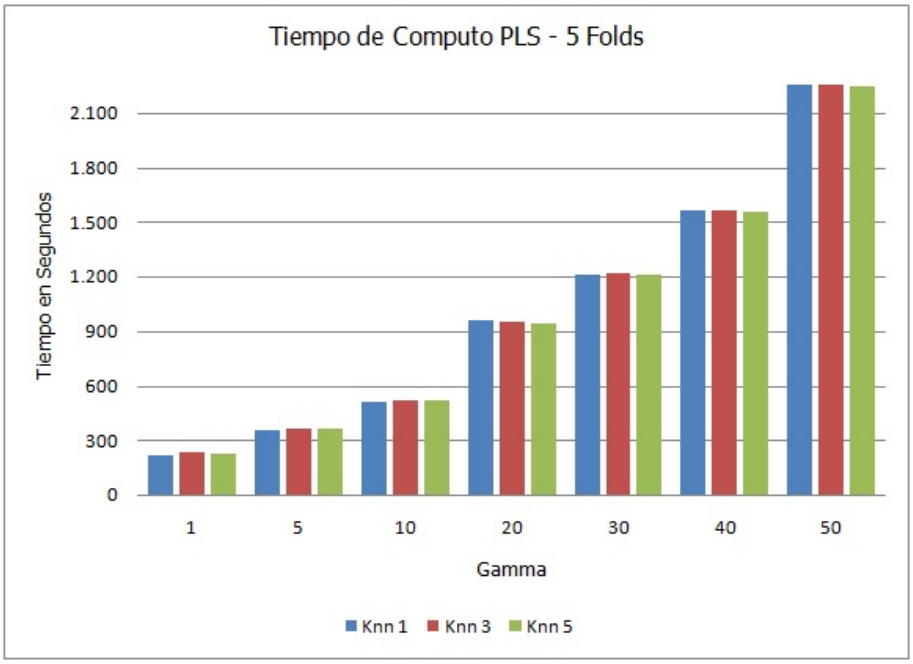
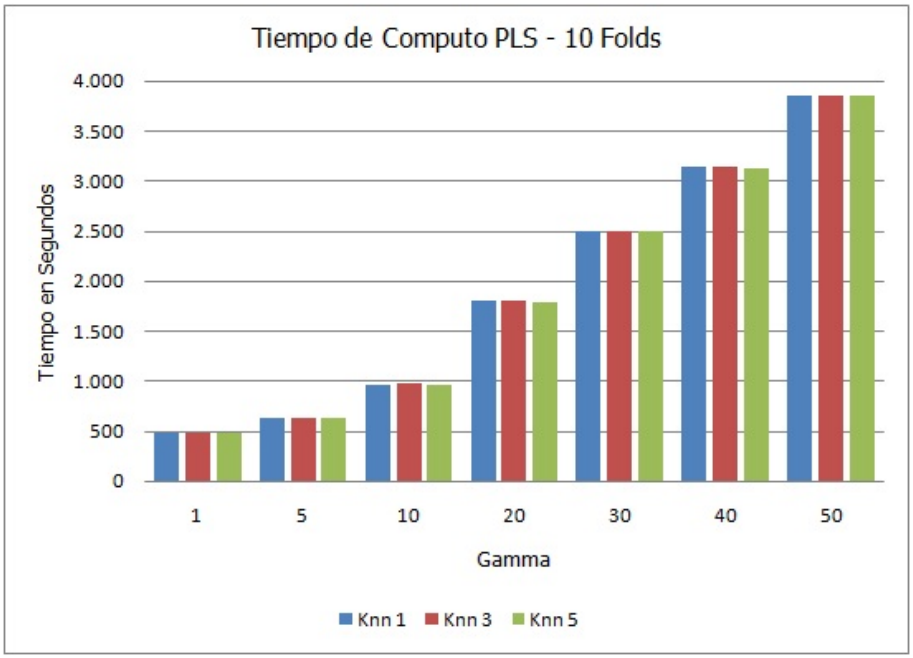


Figura 14. Tiempo de Computo de PLS-DA con 10 Kfolds



Siguiendo nuestro análisis en la Figura 15 y la Figura 16 se presenta las matrices de confusión promedio resultantes de aplicar PLS-DA y posteriormente Knn con parámetros $\gamma = 50$ y $k = 1$ sobre los dos Kfolds, esto es el promedio posición a posición de las matrices de confusión de cada fold. En ellas se puede ver como si bien el método es altamente efectivo como replicaba anteriormente tanto la Figura 11, como la Figura 12, el mismo presenta tiene problemas marcados para reconocer

dígitos ambiguos como el ”4” y el ”9” o el ”3” y el ”8”.

Figura 15. Matriz de Confusión k(1) PLS-DA con 5 Kfolds

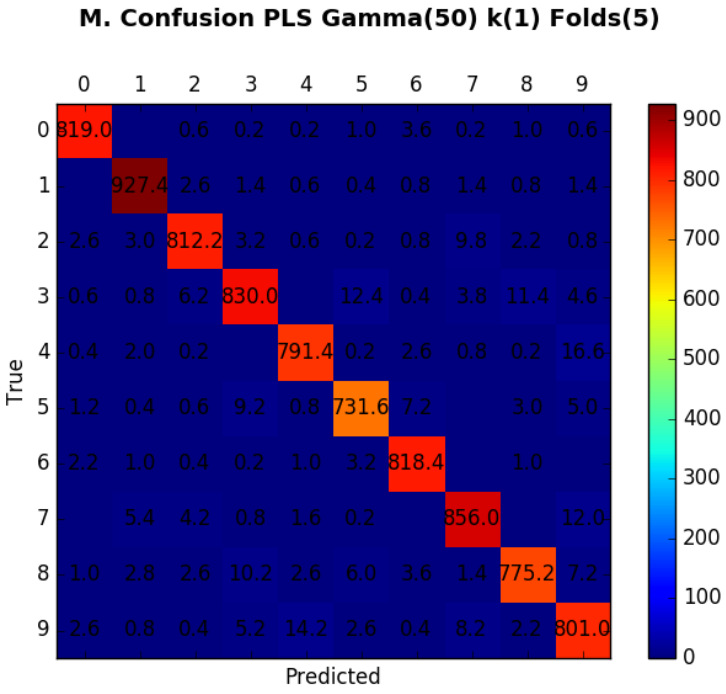
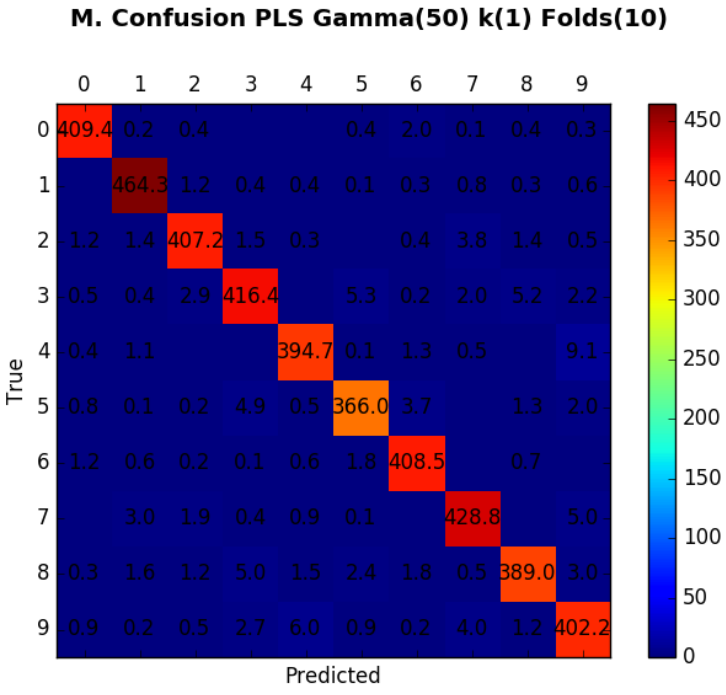


Figura 16. Matriz de Confusión k(1) PLS-DA con 10 Kfolds



En este experimento, seguimos observando que Knn presenta mayor efectividad para $k = 1$, ademas de seguir la misma tendencia que en el espacio original. Como conclusión se puede rescatar que si bien los tiempos de computo para realizar el reconocimiento del dígito con Knn, previamente aplicado PLS-DA con $\gamma = 50$ es similar al que tiene Knn en su versión primitiva, esto es sin pre procesar, lo que a priori parecería descartar PLS-DA ya que anula su principal propósito, seria cuando menos una idea errónea ya que si bien el costo de obtener la matriz de la transformación es alto, esta en un caso particular podría estar pre calculada sobre nuestra base actual, y de esta manera el tiempo de computo de nuestra predicción seria prácticamente despreciable.
También se concluye que hay una relación directa entre nuestros indices de efectividad y γ, nuestra suposición anterior de que la efectividad con respecto a γ parecía estacionarse o estar cerca un punto critico, es algo que no queda completamente claro, por lo que se verá y suscito otro experimento.

Por ultimo concluimos que PLS-DA es un método altamente efectivo para el pre procesamiento de imágenes, ya que presenta una gran efectividad en γ cercanos a 50 y aunque no es despreciable su tiempo de cómputo, este puede ser solucionado parcialmente, de forma tal que sea transparente al usuario.

4.5. Tamaño

4.5.1. Objetivo del experimento

El objetivo del experimento está centrado en medir cómo varía la efectividad de los métodos ante una menor cantidad de elementos para usar de entrenamiento. En particular, para parámetros α y γ fijos para PCA y PLS respectivamente, cómo afecta la cantidad de muestras y la cantidad de vecinos que toma el KNN.

4.5.2. Desarrollo

Definimos los casos de experimento variando 3 parámetros:

- KNN: Probamos utilizando 1, 3 y 5 para los vecinos más cercanos. En el primer experimento, estos fueron los parámetros que obtuvieron los mejores resultados. Si bien no podemos garantizar que sean los óptimos para cada caso que estamos testeando, son buenos candidatos para observar el efecto que tiene variar el tamaño de la base.
- Folds: Para validar los resultados, utilizamos k-fold cross validation. En este caso utilizamos 5-fold y 10-fold por dos razones en especial. Primero, para mantener consistencia con el resto de los experimentos . Segundo, para analizar que sucede cuando la base de testeo compone un 10 % o 20 % de las muestras.
- Tamaño: El eje central del experimento. Por limitaciones temporales trabajamos sólo con 3 tamaños distintos. Teniendo en cuenta esa restricción nos propusimos buscar bases que estén lo suficientemente separadas entre sí y del

total de manera de obtener resultados diferentes y significativos. Decidimos trabajar con 5000, 10000 y 30000 muestras.

Dado que ambos métodos calculan y utilizan matrices de covarianza, mientras más muestras observemos más precisas son las matrices. Esperamos que al reducir la cantidad de muestras, el desempeño se degrade. Consideramos que con menos muestras, es más difícil caracterizar y reorganizar el espacio correspondiente para la transformación de PCA y que con bases más chicas, PLS-DA tendrá mejores resultados

4.5.3. Resultados

Los resultados obtenidos fueron los siguientes:

Tabla 1. Métricas para PCA con 5-fold

KNN	Tamaño	HitRate	Recall Mínimo	Recall Promedio	Precisión Mínimo	Precisión Promedio
1	5000	0.937200	0.865200	0.935686	0.851803	0.936464
	10000	0.954900	0.914027	0.954330	0.905401	0.954914
	30000	0.968867	0.940294	0.968546	0.936665	0.968842
	42000	0.972381	0.953049	0.972177	0.944309	0.972293
3	5000	0.901200	0.757292	0.899673	0.688330	0.911512
	10000	0.926500	0.810508	0.925710	0.760181	0.933071
	30000	0.952700	0.890434	0.952393	0.842524	0.954957
	42000	0.958286	0.906826	0.957982	0.860986	0.959972
5	5000	0.857000	0.640541	0.854513	0.584516	0.882361
	10000	0.892300	0.709915	0.891067	0.670450	0.908164
	30000	0.932367	0.839487	0.931969	0.774533	0.938251
	42000	0.939429	0.860910	0.938953	0.795968	0.944111

Tabla 2. Métricas para PLS-DA con 5-fold

KNN	Tamaño	HitRate	Recall Mínimo	Recall Promedio	Precisión Mínimo	Precisión Promedio
1	5000	0.938800	0.856395	0.937041	0.863020	0.937822
	10000	0.954500	0.911166	0.954027	0.902437	0.954456
	30000	0.968733	0.934932	0.968380	0.936118	0.968672
	42000	0.971690	0.951043	0.971496	0.943044	0.971552
3	5000	0.905200	0.774312	0.903537	0.698950	0.915061
	10000	0.928400	0.825681	0.927801	0.772203	0.934283
	30000	0.952733	0.892469	0.952534	0.844658	0.954861
	42000	0.957548	0.905249	0.957306	0.861375	0.959156
5	5000	0.861800	0.656536	0.859417	0.591247	0.886227
	10000	0.896200	0.739123	0.895152	0.678514	0.911207
	30000	0.930500	0.838637	0.930213	0.769933	0.936756
	42000	0.938524	0.857598	0.938193	0.795422	0.943335

Tabla 3. Métricas para PCA con 10-fold

KNN	Tamaño	HitRate	Recall Mínimo	Recall Promedio	Precisión Mínimo	Precisión Promedio
1	5000	0.942000	0.856530	0.940793	0.859931	0.941815
	10000	0.958700	0.910902	0.958531	0.907094	0.958774
	30000	0.969367	0.941618	0.968784	0.935616	0.969297
	42000	0.973357	0.948814	0.973122	0.946571	0.973260
3	5000	0.910600	0.763369	0.909327	0.708244	0.919466
	10000	0.930200	0.825054	0.929570	0.768452	0.935958
	30000	0.953933	0.896581	0.953455	0.854112	0.955884
	42000	0.959643	0.909216	0.959339	0.865888	0.961220
5	5000	0.870000	0.651086	0.867921	0.602756	0.892390
	10000	0.898000	0.723926	0.897323	0.679806	0.912642
	30000	0.935033	0.852194	0.934422	0.789478	0.940106
	42000	0.941952	0.861186	0.941492	0.806069	0.946103

Tabla 4. Métricas para PLS-DA con 10-fold

KNN	Tamaño	HitRate	Recall Mínimo	Recall Promedio	Precisión Mínimo	Precisión Promedio
1	5000	0.895000	0.764546	0.891977	0.776971	0.892478
	10000	0.902200	0.813300	0.900422	0.812067	0.900592
	30000	0.918067	0.843291	0.916470	0.854600	0.916387
	42000	0.972976	0.947851	0.972705	0.944675	0.972897
3	5000	0.853600	0.664354	0.851421	0.628261	0.869496
	10000	0.870700	0.718619	0.869025	0.679837	0.883169
	30000	0.892100	0.754323	0.890757	0.741748	0.898523
	42000	0.959595	0.906718	0.959374	0.868388	0.961017
5	5000	0.810400	0.564270	0.807907	0.551766	0.845352
	10000	0.829700	0.634937	0.827416	0.593469	0.858175
	30000	0.863000	0.686172	0.861568	0.671301	0.878779
	42000	0.941238	0.863463	0.940821	0.802126	0.945534

4.5.4. Discusión de los resultados

Para cada caso de k-fold y KNN, podemos observar que a medida que el tamaño decrece la performance empeora. Sin embargo, aún para bases de 5000 muestras (un tamaño considerablemente menor a la base original) hay casos donde el hitrate, precisión y recall promedio están por encima del 90 %. Estos resultados incentivaron el siguiente experimento donde observamos que pasa con bases aún más chicas.

Además parece mantenerse la tendencia observada anteriormente con los valores de KNN. A medida que aumentan, la efectividad de los métodos decrece.

Se dio por otro lado una situación que no esperábamos. Corriendo sobre 5-fold la diferencia entre PLS Y PCA es mínima, ambas implementaciones se comportan de modo similar. Para el caso de 10-fold, hay una diferencia cercana al 5 % que favorece PLS-DA sobre PCA. Lo opuesto a lo que habíamos predicho.

Una posible explicación podría encontrarse en la naturaleza de ambos algoritmos. Al ser supervisado y considerar la etiqueta de cada dígito, es posible que sea más sensible a reducciones de tamaño dado que trabaja con subconjuntos más pequeños que PCA.

4.6. Tamaño (continuación)

4.6.1. Objetivo del experimento

En el experimento anterior observamos que para 5-fold y tomando el vecino más cercano, incluso la base más chica que elegimos (5000 elementos) mantenía un hitrate superior al 90 %. El objetivo de este experimento es profundizar sobre esas observaciones, viendo como se comportan tanto PLS Y PCA con bases extremadamente chicas manteniendo las otras condiciones estables.

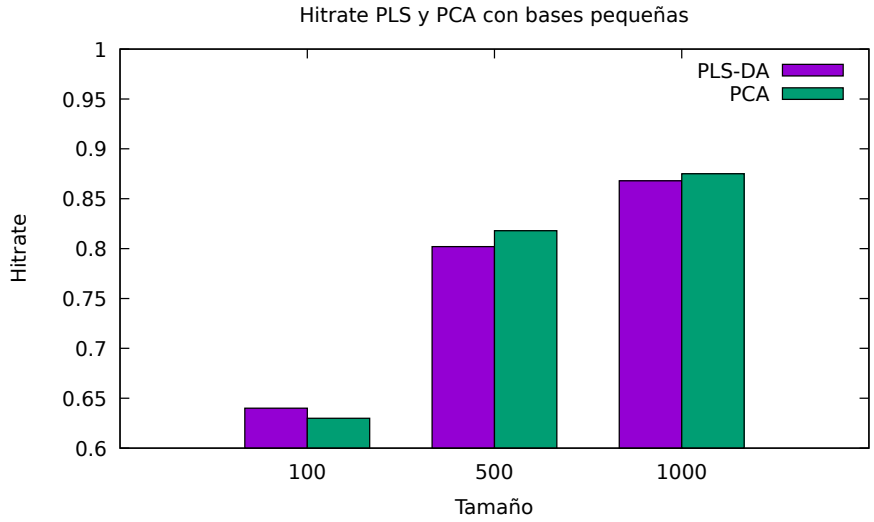
4.6.2. Desarrollo

Como queremos replicar las condiciones del experimento anterior, vamos a mantener el α y γ de PCA y PLS en 50. Además, mantenemos las condiciones de KNN y K-fold que desataron esta experimentación 5-fold y el vecino más cercano. Para las bases, buscamos tomar tamaños menores a los del primer experimento, pero con suficientes elementos para que tenga sentido utilizar los métodos. Finalmente elegimos bases de 1000, 500 y 100 elementos. Con esos parámetros, analizamos el hitrate.

4.6.3. Resultados

Los resultados obtenidos fueron los siguientes:

Figura 17. Hitrate de ambos métodos según base



4.6.4. Discusión de los resultados

Estos resultados nos informan de la velocidad con la cuál degrada la efectividad de los métodos. Si bien para los casos de 500 y 1000 muestras el hitrate ronda valores cercanos al 80 %, se desploma abruptamente para 100 muestras. cayendo por debajo

del 65 %. La diferencia entre bases de 100 y 500 es similar a la que hay entre 500 y 5000.

Además observamos que si bien hay pequeñas diferencias entre PLS y PCA, se comportan relativamente de la misma manera. No se puede concluir en términos de efectividad que un método supere claramente al otro.

4.7. α y γ grandes.

4.7.1. Desarrollo

Este experimento se ideó e base a los experimentos anteriores, sobre los métodos PCA y PLS, en los que quedó pendiente concluir cuales eran los valores α y γ óptimos. En este nuevo experimento ampliaremos las observaciones realizadas hasta ahora, pero evaluando nuevos valores de α y γ hasta llegar a 150 en ambos casos. Nos detuvimos en 150 porque observamos que el hitrate ya había llegado a un máximo y a partir de entonces descendía levemente.

4.7.2. Resultados

En primer lugar, en la Figura 18 y Figura 19 se pueden ver la evolución ampliada del hitrate con 5 Kfolds para PCA y PLS-da respectivamente. Luego en la Figura 20, Figura 21, Figura 22 y Figura 23 los datos de efectividad detallados, para k=1.

Figura 18. Hitrate ampliado de PCA con 5 Kfolds

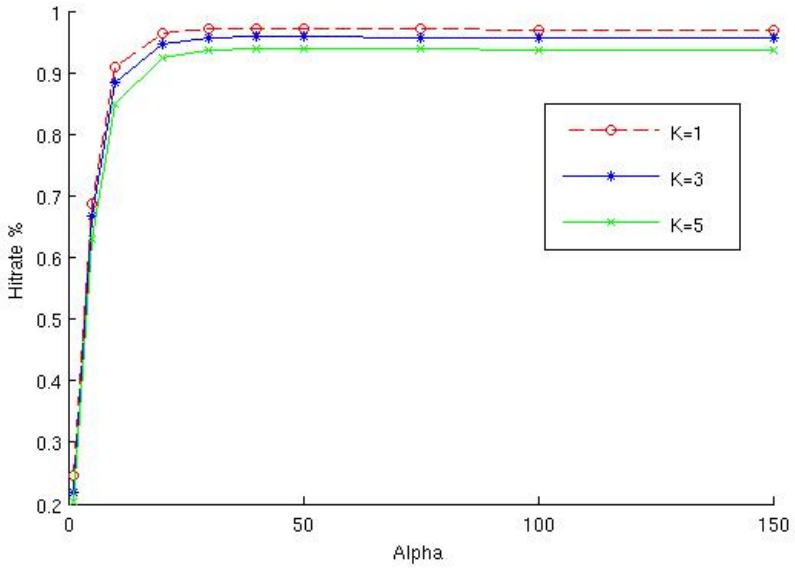


Figura 19. Hitrate ampliado de PLS con 5 Kfolds

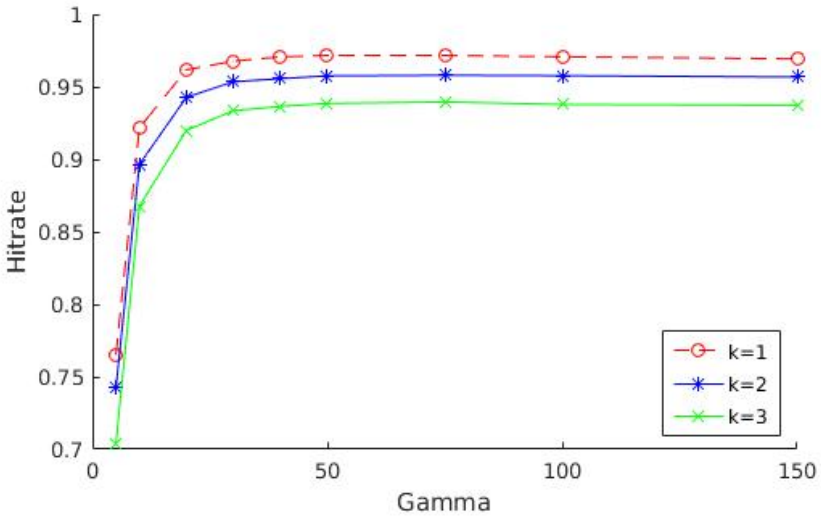


Figura 20. Efectividad ampliada de PCA con 5 Kfolds, con k=1

Alpha	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.245286	0.106383	0.238065	0.104752	0.238625
5	0.687714	0.531089	0.683826	0.534819	0.683515
10	0.909524	0.823155	0.908432	0.814193	0.908387
20	0.963238	0.932885	0.962941	0.921434	0.962974
30	0.971810	0.950213	0.971604	0.943140	0.971639
40	0.972286	0.951589	0.972058	0.947021	0.972110
50	0.972381	0.953049	0.972177	0.944309	0.972293
75	0.970881	0.945055	0.970616	0.943935	0.970825
100	0.970000	0.942646	0.969693	0.942477	0.969968
150	0.968667	0.936216	0.968303	0.940818	0.968737

Figura 21. Efectividad ampliada de PCA con 10 Kfolds, con k=1

K	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.242238	0.095239	0.234933	0.096988	0.234296
5	0.689024	0.535112	0.685154	0.535777	0.684856
10	0.911262	0.826707	0.910191	0.810956	0.910258
20	0.964857	0.930573	0.964560	0.924620	0.964532
30	0.972857	0.945441	0.972625	0.944068	0.972711
40	0.973000	0.948053	0.972745	0.946130	0.972921
50	0.973357	0.948814	0.973122	0.946571	0.973260
75	0.972476	0.944088	0.972175	0.945567	0.972418
100	0.971429	0.942324	0.971077	0.943267	0.971379
150	0.970167	0.937886	0.969757	0.942900	0.970208

Figura 22. Efectividad ampliada de PLS con 5 Kfolds, con k=1

Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.259238	0.117798	0.253116	0.121815	0.252892
5	0.765000	0.599659	0.761874	0.608680	0.761033
10	0.921690	0.863092	0.920564	0.862399	0.920694
20	0.961714	0.932255	0.961376	0.918210	0.961428
30	0.967667	0.944930	0.967387	0.936697	0.967440
40	0.970595	0.948924	0.970384	0.939660	0.970427
50	0.971690	0.951043	0.971496	0.943044	0.971552
75	0.971619	0.948253	0.971349	0.945390	0.971558
100	0.970690	0.943127	0.970418	0.943336	0.970604
150	0.969262	0.938699	0.968907	0.940674	0.969279

Figura 23. Efectividad ampliada de PLS con 10 Kfolds, con k=1

Gamma	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.259619	0.111044	0.253519	0.108345	0.254218
5	0.766048	0.601404	0.762863	0.608328	0.762161
10	0.923738	0.859979	0.922640	0.868266	0.922618
20	0.961905	0.927471	0.961528	0.920393	0.961618
30	0.968429	0.937501	0.968115	0.933045	0.968215
40	0.972214	0.944891	0.971963	0.940409	0.972075
50	0.972976	0.947851	0.972705	0.944675	0.972897
75	0.972595	0.944724	0.972265	0.944372	0.972514
100	0.972357	0.944188	0.972035	0.945153	0.972275
150	0.970929	0.939875	0.970538	0.944581	0.970918

4.7.3. Discusión de los resultados

En base a los resultados, se puede concluir que tanto en el método PCA como en PLS, el mayor hitrate se consigue en valores de α y γ cercanos a 50. A partir de entonces, la efectividad empieza a descender muy lentamente.

De forma intuitiva, se puede inferir que a medida que se aumenten estos parámetros, lo resultados deberían ser cada vez mas similares a los obtenidos al utilizar el método knn sin ninguna transformación del espacio previa.

4.8. Knn Posicional

4.8.1. Desarrollo

En este experimento se planteo ver que mejoras podía tener efectuar ”Knn Versión 2” o ”Knn Posicional”, al que vamos a llamar resumidamente ”KnnV2”, este es un algoritmo que funciona de forma similar a ”Knn”, con la diferencia de que implementa lo que llamamos una ”Moda Posicional”, en esta a cada elemento dentro de los k mas cercanos se le asigna un puntaje en función de cuan ”cerca” estaba del elemento a procesar, llamesmolo β, esto se hace a través de rangos determinados por selectos escalares y la norma dos del elemento a procesar, como ultimo paso se elije como Moda Posicional al de mayor puntaje entre los k mas cercanos. Este algoritmo surgió no solo de ver el mal desempeño de Knn para k ”grandes”, sino del hecho de no dar el mismo peso a elementos distantes.

No se va a abordar el detalle de este algoritmo ya que no es el objetivo de este experimento, y que se adjunto su implementan en C++ en el Apéndice Código Fuente.

4.8.2. Resultados

Tanto en la Figura 24, como en la Figura 25 se puede observar que a medida que aumenta k la efectividad de nuestro método decrece al igual que en Knn, pero en este caso lo hace suavemente, ademas hay mejoras para k 3 y 5 en la efectividad con respecto a $k = 1$ como se observa en la Figura 25, otro punto notable es que los otros indicadores de efectividad si también siguen la tendencia suave pero descendente algo que no ocurría en Knn, ya que para k "grandes" estos decrecían drásticamente y sin un patrón fijo entre menor y promedio.

Figura 24. Efectividad de knnV2 con 5 Kfolds

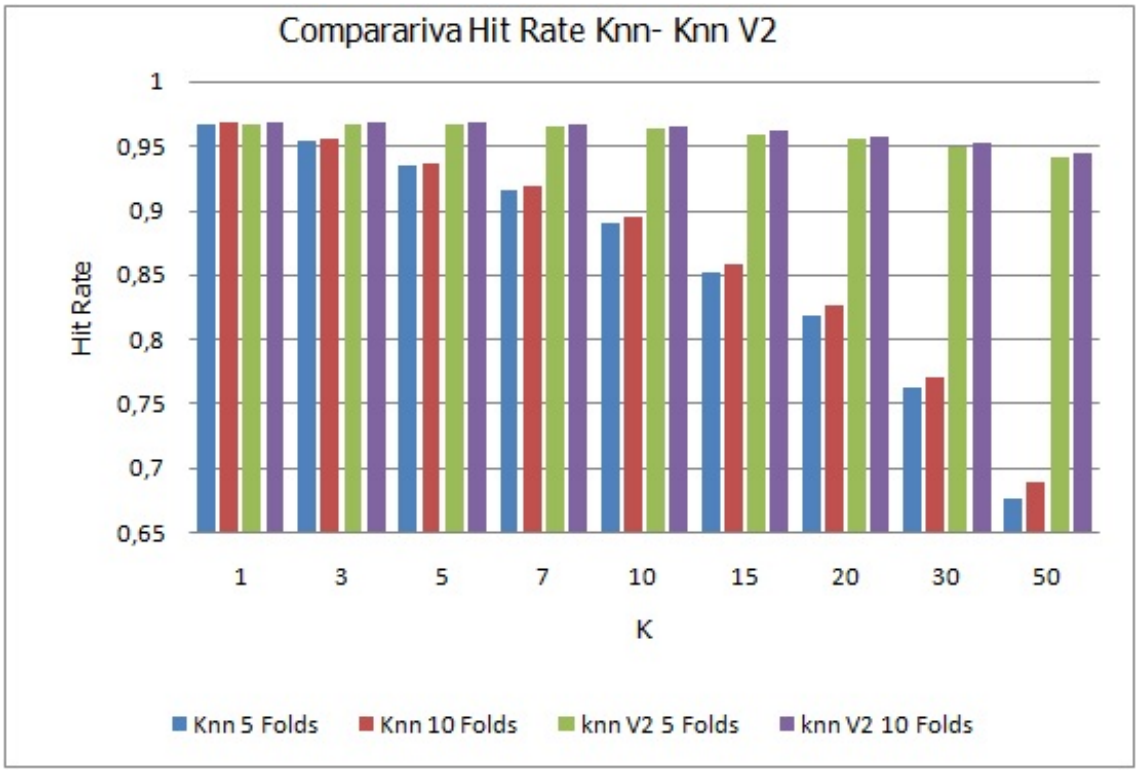
5 Folds – KnnV2					
K	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.966833	0.930271	0.966402	0.937964	0.967011
3	0.966762	0.920892	0.966278	0.944302	0.967243
5	0.965929	0.916799	0.965416	0.940727	0.966518
7	0.965071	0.918290	0.964561	0.935287	0.965862
10	0.962714	0.909988	0.962133	0.931888	0.963740
15	0.959190	0.905064	0.958547	0.920810	0.960568
20	0.955214	0.896671	0.954525	0.914024	0.956804
30	0.949333	0.884598	0.948583	0.900344	0.951421
50	0.941357	0.868159	0.940519	0.877198	0.944431

Figura 25. Efectividad de knnV2 con 10 Kfolds

10 Folds – KnnV2					
K	Hitrate	Menor Recall	Recall Promedio	Menor Precision	Precision Promedio
1	0.967690	0.928948	0.967182	0.939080	0.967853
3	0.968333	0.924245	0.967827	0.945107	0.968712
5	0.967810	0.923704	0.967307	0.942198	0.968261
7	0.966524	0.921585	0.965984	0.939068	0.967239
10	0.964738	0.914966	0.964155	0.935323	0.965652
15	0.961238	0.907405	0.960631	0.927682	0.962360
20	0.957595	0.898847	0.956904	0.919060	0.959060
30	0.952190	0.887712	0.951500	0.902657	0.954219
50	0.944095	0.869535	0.943288	0.881794	0.947034

Aquí en la figura Figura 26 se ve claramente reflejado como decae la efectividad en Knn, a medida que aumenta k, mientras que en KnnV2 se mantiene mas constante.

Figura 26. Comparación HitRate Knn/KnnV2



Siguiendo con el análisis tanto en la Figura 27, como la Figura 28, se evidencia que no hubo mejoras significativas en el reconocimiento de dígitos ambiguos, esto era algo esperable ya que los dígitos ambiguos precisamente tienen como característica estar ”cerca”.

Figura 27. Matriz de Confusión KnnV2 k(3) con 10 Kfolds

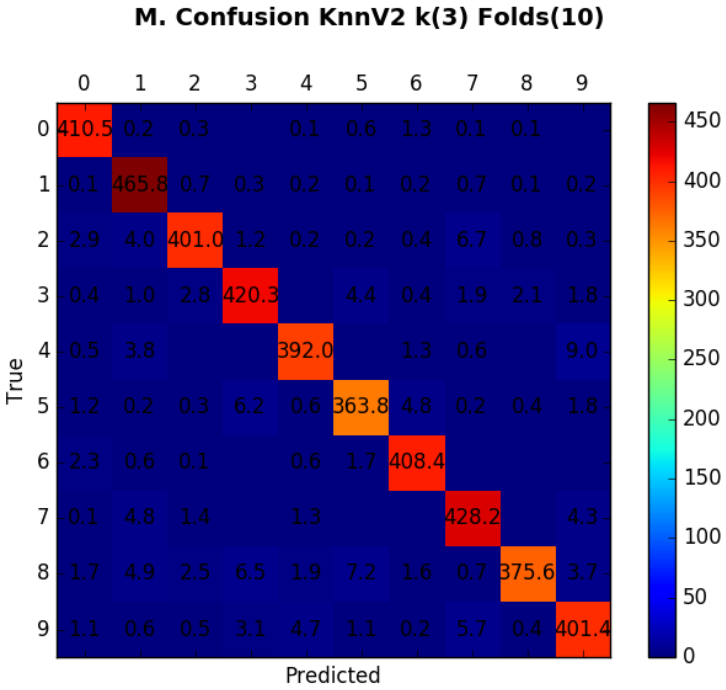
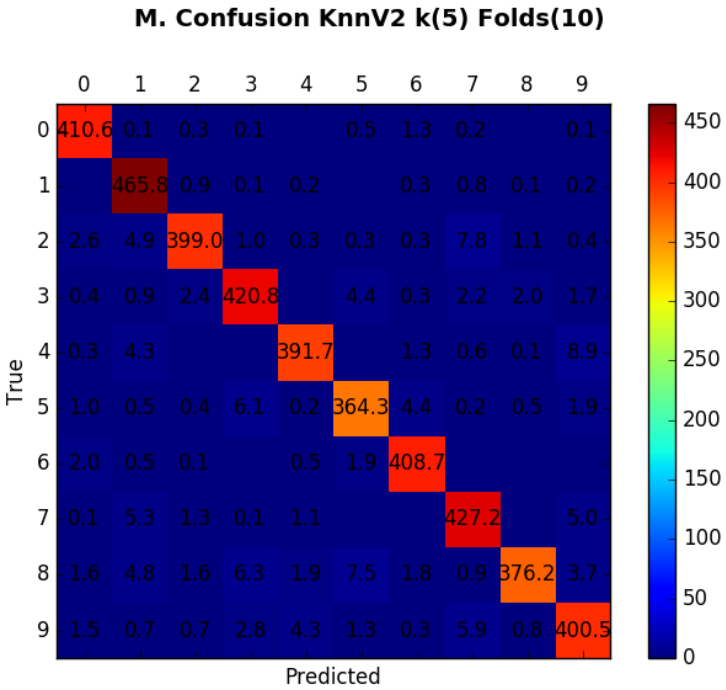


Figura 28. Matriz de Confusión KnnV2 k(5) con 10 Kfolds



Como conclusión de este experimento notamos que si bien no hallamos mejoras en función de lograr un HitRate notablemente superior, los buenos resultados que

obtuvimos para k altos y las mejoras en todo sentido de Efectividad, auspician que este Algoritmo de Knn podría ser ampliamente mejorado, colocando en primer medida puntajes óptimos en función de los elementos a procesar, lo que auspiciaría mejoras mas notorias. Algo que este algoritmo no presenta ya que no era el objetivo del trabajo encontrar el optimo, sino ver como influía, por lo que puede ser un trabajo a futuro nada despreciable de realizar.

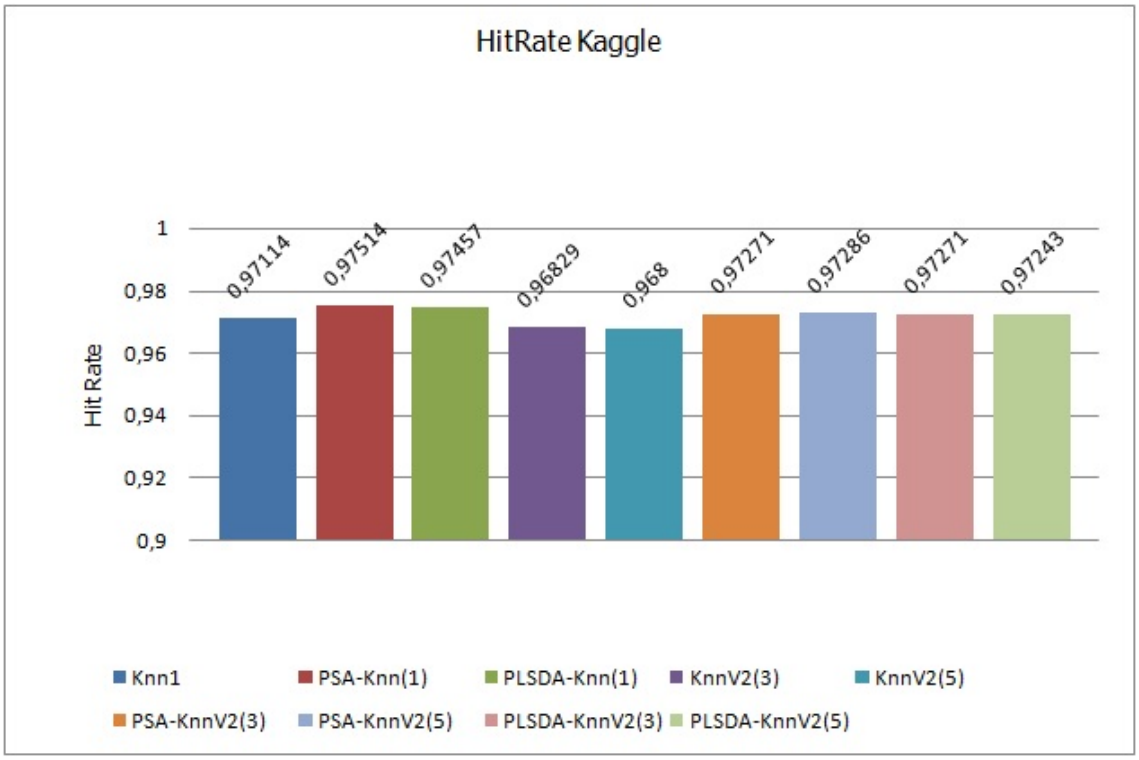
4.9. Kaggle

Por ultimo alentados por la cátedra y para obtener un feedback real, de nuestros métodos participamos de la competencia "Digit Recognizer"llevada a cabo por "www.kaggle.com", en la cual se ponen a prueba técnicas de Reconocimiento de Patrones como las desarrolladas en este trabajo para el reconocimiento de dígitos. A continuación en la Figura 29 presentamos los resultados obtenidos para algunos de nuestros métodos, los cuales elegimos en función de su Efectividad.

Los métodos elegidos fueron los siguientes:

1. Knn1, Knn con $k = 1$.
2. PSA-Knn(1), PSA con $\alpha = 50$, Knn con $k = 1$.
3. PLSDA-Knn(1), PLS-DA con $\gamma = 50$, Knn con $k = 1$.
4. KnnV2(3), PSA con $\alpha = 50$, KnnV2 con $k = 3$.
5. KnnV2(5), PSA con $\alpha = 50$, KnnV2 con $k = 5$.
6. PSA-KnnV2(3), PSA con $\alpha = 50$, KnnV2 con $k = 3$.
7. PSA-KnnV2(5), PSA con $\alpha = 50$, KnnV2 con $k = 5$.
8. PLSDA-KnnV2(3), PLS-DA con $\gamma = 50$, KnnV2 con $k = 3$.
9. PLSDA-KnnV2(5),PLS-DA con $\gamma = 50$, KnnV2 con $k = 5$.

Figura 29. Comparativa Hit Rate - Digit Recognizer



4.10. Conclusiones Finales

Luego de haber realizado este trabajo pudimos ver que reconocer patrones no es un problema que tenga una resolución trivial, y que cada problema particular puede tener una solución adecuada distinta, sin embargo en este trabajo se analizaron los métodos usados bajo esquemas vectoriales, con los que se pueden modelar gran cantidad de problemas de reconocimiento de patrones y por lo tanto obtuvimos múltiples conclusiones que son aplicables a los esquemas descriptos previamente. De estas destacamos:

1. El caudal de datos para ”entrenar” los métodos trabajados es de vital importancia, y presenta una correlación directa con la efectividad de los mismos. Por lo que la conformación de bases de datos sobre los cuales operaran posteriormente los métodos es aunque oculto el paso mas importante de todo método de reconocimiento de patrones.
   Así mismo dado que no se conoce a priori los parámetros óptimos a usar en nuestros métodos, esto es en sí mismo gran parte de la fase de entrenamiento, es de vital importancia no cometer errores de medición o sacar conclusiones erróneas en esta fase, es por esto que cobra prioridad el método de validación de resultados a utilizar, en este caso ”K-fold cross validation”, del cual la elección del ”K” debe estar correctamente ligada a la cantidad de datos disponibles.

2. Los métodos de pre-procesamiento usados adecuadamente mostraron ser de gran utilidad para bajar los tiempos de cómputo, o inclusive prácticamente nulificarlos al amortizarlos, ya que uno podría mantener dentro de su estructura la matriz correspondiente a la transformación buscada y de esa forma procesar nuestros elementos a un costo muy bajo. Asimismo también mostraron mejoras en la efectividad, pese a no obtener los parámetros ”óptimos” para estos métodos.

3. En los algoritmos para implementar los métodos se debe buscar en los casos que se pueda, ya sea recurriendo al álgebra u otros métodos conocidos, reducir las operaciones sobre elementos de gran tamaño, dado que estas pueden derivar en enormes tiempos de cómputo, al ser la mayoría de los métodos presentados muy sensibles a la dimensión de los elementos a procesar.

4. El método ”knn” si bien es muy efectivo para tratar con problemas como los presentados en este trabajo, presenta amplio lugar a mejoras, dado que no solo presenta una gran sensibilidad a la dimensión de los elementos a tratar (algo que se puede solucionar con pre procesamiento) sino que también la moda que aplica presenta problemas serios a la hora de reconocer elementos ambiguos, algo que con dígitos no parece ser un gran problema, dado que mayoritariamente son distinguibles, pero en escenarios donde lo que se modela son otros objetos menos notorios, como células por ejemplo, la detección de elementos visualmente ambiguos cobra vital importancia.

5. Los métodos de búsqueda de autovalores y autovectores como método de la potencia, sobre el cual experimentamos en este trabajo, aunque parecen ser simplemente un resultado matemático, son altamente relevantes para los métodos de reconocimiento de patrones, tanto para los trabajados aquí como para muchos otros. Por lo que poseer métodos confiables y ágiles para obtener autovalores y autovectores es de gran importancia.

5. Apéndices

5.1. Enunciado

Métodos Numéricos
Segundo Cuatrimestre 2016
Trabajo Práctico 2

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

CSI:DC

Introducción

En la actualidad, las ciencias de la computación son útiles para resolver un abanico muy abarcativo de problemas. En los últimos años la inmersión de herramientas computacionales para la resolución de problemas de índole criminalística ha sido abrumadora, seguramente motivada por la interminable lista de películas y series *hollywoodenses* al respecto. Como muchas otras modas, esta también desembarcó en Argentina y son varios los interesados en manejar estas tecnologías para mostrarse como líderes del segmento.

En este contexto, una entidad que mantendremos en el anonimato ha mostrado interés en que un grupo selecto de personas relacionadas con el DC desarrolle un sistema que pueda reconocer texto manuscrito. Sin embargo, para lograr entablar relaciones con dicha entidad, se ha acordado entre ambas partes la entrega de un módulo de reconocimiento de dígitos como prueba de concepto. El desarrollo de dicho módulo es de extrema importancia dado que es el puntapié inicial para comenzar las relaciones, sumado a que será una parte fundamental del software completo que se desarrolle potencialmente en un futuro.

Con plena confianza, se decidió que sean los alumnos de Métodos Numéricos quienes lleven a cabo el desarrollo del proyecto.

Objetivos y Metodología

Como instancias de entrenamiento, se tiene una base de datos de n imágenes de $M \times M$ píxeles, las cuales se encuentran etiquetadas con el dígito, 0-9, al que corresponden. Definimos como $m = M \times M$ al número total de píxeles de una imagen. Asumiremos también que las imágenes se encuentran en escala de grises (cada pixel corresponde a un valor entre 0 y 255, indicando la intensidad del mismo) y que el etiquetado no contiene errores. El objetivo del trabajo consiste en utilizar la información de la base de datos para, dada una nueva imagen de un dígito sin etiquetar, determinar a cuál corresponde teniendo en cuenta factores de calidad y tiempo de ejecución requeridos.

Una primera aproximación es utilizar el conocido algoritmo de k *Vecinos más Cercanos* (kNN, por su nombre en inglés). En su versión más simple, este algoritmo considera a cada objeto de la base de entrenamiento como un punto en el espacio, para el cual se conoce a qué clase corresponde (en nuestro caso, qué dígito es). Luego, al obtener un nuevo objeto que se busca clasificar, simplemente se buscan los k vecinos más cercanos y se le asigna la clase que posea el mayor número de repeticiones dentro de ese subconjunto, es decir, la moda. Con este objetivo, podemos representar a cada imagen de nuestra base de datos como un vector $x_i \in \mathbb{R}^m$, con $i = 1, \ldots, n$, y de forma análoga interpretar las imágenes a clasificar mediante el algoritmo kNN.

Sin embargo, el algoritmo del vecino más cercano es sensible a la dimensión de los objetos a considerar y, aún aplicando técnicas de preprocesamiento, puede resultar muy costoso de computar. Teniendo en cuenta esto, una alternativa interesante de preprocesamiento es buscar reducir la cantidad de dimensiones de las muestras para trabajar con una cantidad de

variables más acotada y, simultáneamente, buscando que las nuevas variables tengan información representativa para clasificar los objetos de la base de entrada. En esta dirección, consideraremos dos métodos de reducción de dimensiones: *Análisis de Componentes Principales* (PCA, por su sigla en inglés) y *Análisis discriminante con cuadrados mínimos parciales* (PLS-DA, por su sigla en inglés).

PCA

El método de análisis de componentes principales consiste en lo siguiente. Para $i = 1, \ldots, n$, recordar que $x_i \in \mathbb{R}^m$ corresponde a 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

$$M = X^t X$$

a la matriz de covarianza de la muestra X. Siendo v_j el autovector de M asociado al j-ésimo autovalor al ser ordenados por su valor absoluto , definimos para $i = 1, \ldots, n$ la *transformación característica* del dígito x_i como el vector $\mathbf{tc_{PCA}}(x_i) = (v_1^t\, x_i, v_2^t\, x_i, \ldots, v_\alpha^t\, x_i) \in \mathbb{R}^\alpha$, donde $\alpha \in \{1, \ldots, m\}$ es un parámetro de la implementación. Este proceso corresponde a extraer las α primeras *componentes principales* de cada imagen. La intención es que $\mathbf{tc_{PCA}}(x_i)$ resuma la información más relevante de la imagen, descartando los detalles o las zonas que no aportan rasgos distintivos.

PLS-DA

En el caso de PLS-DA, la idea es similar al método de componentes principales con la diferencia de la información original que se utiliza para realizar la transformación. En este caso, en vez de utilizar solamente las imágenes de entrenamiento, se utilizan también las clases a las que pertenecen dichas imágenes para influenciar la transformación característica. Definimos X de manera análoga al algoritmo PCA. $preY \in \mathbb{R}^{n \times 10}$ como una matriz que tiene un 1 en la posición $preY_{i,j}$ si la muestra i de la base de entrenamiento corresponde al dígito $j - 1$, y -1 en el caso contrario. Por último, definimos Y como la matriz resultante de tomar $preY$, sustraer el promedio de todas las filas a cada una de ellas, y dividir por la raíz cuadrada de la cantidad de muestras menos 1. Luego, se utiliza el siguiente pseudocódigo para generar los vectores w_j.

Data: X,Y,γ
Result: $w_1 \ldots w_j$
for $i = 1 \ldots \gamma$ **do**
  definir M_i como $X^t Y Y^t X$;
  calcular w_i como el autovector asociado al mayor autovalor de M_i;
  normalizar w_i utilizando norma 2; definir t_i como $X w_i$;
  normalizar t_i utilizando norma 2; actualizar X como $X - t_i t_i^t X$;
  actualizar Y como $Y - t_i t_i^t Y$;
end

En el anterior pseudocódigo, γ es un parámetro para controlar la cantidad de dimensiones a utilizar. Una vez obtenidos los vectores w_j, la transformación característica de este método se define de forma análoga como $\mathbf{tc_{PLS}}(x_i) = (w_1^t\, x_i, w_2^t\, x_i, \ldots, w_\gamma^t\, x_i) \in \mathbb{R}^\gamma$

Como se mencionó anteriormente, los métodos PCA y PLS-DA recientemente presentados no son métodos de clasificación en sí, sino que sirven para realizar una transformación de los datos de entrada. Dada una nueva imagen x de un dígito, se calcula $\mathbf{tc_m}(x)$, siendo tc_m la

transformación característica del método que se esté utilizando, y se compara con $\mathbf{tc_m}(x_i)$, para $i = 1, \ldots, n$, utilizando algún criterio de clasificación adecuado, como por ejemplo kNN.

Finalmente, nos concentramos en la evaluación de los métodos. Dado que necesitamos conocer a qué dígito corresponde una imagen para poder verificar la correctitud de la clasificación, una alternativa es particionar la base de entrenamiento en dos, utilizando una parte de ella en forma completa para el entrenamiento y la restante como test, pudiendo así corroborar la predicción realizada. Sin embargo, realizar toda la experimentación sobre una única partición de la base podría resultar en una incorrecta estimación de parámetros, como por ejemplo el conocido *overfitting*. Luego, se considera la técnica de *cross validation*, en particular el *K-fold cross validation*, para realizar una estimación de los parámetros del modelo que resulte estadísticamente más robusta.

Enunciado

Se pide implementar un programa en `C` o `C++` que lea desde archivos las imágenes de entrenamiento correspondientes a distintos dígitos y que, utilizando los métodos descriptos en la sección anterior, dada una nueva imagen de un dígito determine a qué número pertenece. Para ello, el programa deberá implementar el algoritmo de kNN así como también la reducción de dimensión previa utilizando tanto PCA como PLS-DA.

Con el objetivo de obtener la transformaciones características de cada método, se deberá implementar el método de la potencia con deflación para la estimación de autovalores/autovectores de la matriz de covarianza en el caso de PCA y el algoritmo de deflación de PLS-DA. En este contexto, la factibilidad de aplicar estos métodos es particularmente sensible al tamaño de las imágenes de la base de datos. Por ejemplo, considerar imágenes en escala de grises de 100×100 píxeles implicaría trabajar con matrices de tamaño 10000×10000. Se pide tener en cuenta este factor durante el desarrollo y analizar cómo afecta (si es que lo hace) en el desarrollo del trabajo.

Consideramos la base de datos MNIST, en la versión disponible en *Kaggle*[1]. Esta base contiene un conjunto de datos de entrenamiento de 42.000 dígitos manuscritos en escala de grises y con $M = 28$. A su vez, provee un conjunto de datos de test de 28.000 dígitos, de similares características, pero sin el correspondiente etiquetado. En base a estos datos, se pide separar el procedimiento en dos etapas.

La primera de ellas consiste en realizar un estudio experimental con los métodos propuestos sobre la base de entrenamiento y utilizando la técnica de *K-fold cross validation* mencionada anteriormente. Para esta última tarea en particular, se recomienda leer y utilizar la rutina `cvpartition` provista por MATLAB. Llamamos k a la cantidad de vecinos a considerar en el algoritmo kNN, α a la cantidad de componentes principales a tomar, γ a la cantidad de dimensiones en el método PLS-DA y K a la cantidad de particiones consideradas para el cross-validation.

La calidad de los resultados obtenidos será analizada mediante diferentes métricas. En particular, la métrica más importante que debe reportarse en los experimentos es la tasa de efectividad lograda, es decir, la cantidad dígitos correctamente clasificados respecto a la cantidad total de casos analizados.

Adicionalmente, se mencionan las siguientes métricas para su potencial uso en en el análisis de los experimentos. Se deben utilizar al menos dos de las siguientes métricas, aunque no

[1] www.kaggle.com

necesariamente para todos los experimentos realizados.

Precision: Es una medida de cuántos aciertos relativos tiene un clasificador dentro de una clase particular. Es decir, dada una clase i, la *precision* de dicha clase es $\frac{tp_i}{tp_i+fp_i}$.

En la anterior fórmula, tp_i son los *verdaderos positivos* de la clase i. Es decir, muestras que realmente pertenecían a la clase i y fueron exitosamente identificadas como tales. En contraposición, fp_i son los *falsos positivos* de la clase i. Son aquellas muestras que fueron identificadas como pertenecientes a la clase i cuando realmente no lo eran.

Luego, la *precision* en el caso de un clasificador de muchas clases, se define como el promedio de las *precision* para cada una de las clases.

Recall: Es una medida de qué tan bueno es un clasificador para, dada una clase particular, identificar correctamente a los pertenecientes a esa clase. Es decir, dada una clase i, el *recall* de dicha clase es $\frac{tp_i}{tp_i+fn_i}$.

En la anterior fórmula, fn_i son los *falsos negativos* de la clase i. Es decir, muestras que pertenecían a la clase i pero que fueron identificadas con otra clase.

Luego, el *recall* en el caso de un clasificador de muchas clases, se define como el promedio del *recall* para cada una de las clases.

F1-Score: Dado que *precision* y *recall* son dos medidas importantes que no necesariamente tienen la misma calidad para un mismo clasificador, se define la métrica F1 para medir un compromiso entre el *recall* y la *precision*. La métrica $F1$ se define como $2 * precision * recall/(precision + recall)$.

Kappa de Cohen: Es una medida para indicar cuánto concuerdan dos clasificadores sobre un mismo set de datos. Dicha medida se define como $\kappa = (p_o - p_a)/(1 - p_a)$. Donde p_o es la probabilidad observada de que los dos clasificadores concuerden y p_a es la probabilidad aleatoria de que lo hagan.

Esta métrica puede utilizarse para determinar si el problema contiene ejemplos particularmente complicados, porque por ejemplo ningún clasificador lo reconoce correctamente.

Experimentación

En función de la experimentación se piden como mínimo los siguientes experimentos:

- Analizar la calidad de los resultados obtenidos al combinar PCA y PLS-DA con kNN, para un rango amplio de combinaciones de valores de k, α y γ. Considerar en el análisis también el tiempo de ejecución.
- Analizar la calidad de los resultados obtenidos al combinar PCA y PLS-DA con kNN, para un rango amplio de cantidades de imágenes de entrenamiento. Utilizar desde muy pocas imágenes de entrenamiento hasta todas las disponibles para identificar en que situación se comporta mejor cada uno de los métodos de preprocesamiento.
- Realizar los experimentos de los items anteriores para al menos dos valores distintos de K. Justificar el por qué de la elección de los mismos.
- En base a los resultados obtenidos para ambos métodos, seleccionar aquella combinación de parámetros que se considere la mejor alternativa, con su correspondiente justificación, compararlas entre sí y sugerir un método para su utilización en la práctica.

Finalmente, y con los parámetros seleccionados en la fase experimental, ejecutar el método seleccionado sobre el conjunto de datos de test, utilizando como entrenamiento los 42.000

dígitos comprendidos en el conjunto de entrenamiento. Analizar el tiempo de cómputo requerido. Dado que la cátedra no posee la información sobre a qué dígito corresponde cada imagen (y la idea no es graficar uno por uno y obtenerlo a mano), se sugiere a cada grupo participar en la competencia *Digit Recognizer* actualmente activa en *Kaggle* realizando el/los envíos que consideren apropiados y reportar en el informe los resultados obtenidos.

Puntos opcionales (no obligatorios)

- Mostrar que si tenemos la descomposición $M = U\Sigma V^t$, V es la misma matriz que obtenemos al diagonalizar la matriz de covarianzas.
- Realizar experimentos utilizando dígitos manuscritos creados por el grupo, manteniendo el formato propuesto.[2] Reportar resultados y dificultades encontradas.
- Proponer y/o implementar alguna mejora al algoritmo de kNN.

Programa y formato de archivos

Se deberán entregar los archivos fuentes que contengan la resolución del trabajo práctico. El ejecutable tomará tres parámetros por línea de comando, que serán el nombre del archivo de entrada, el nombre del archivo de salida, y el método a ejecutar (0: kNN, 1: PCA + kNN, 2: PLS-DA + kNN).

Asumimos que la base de datos de imágenes de entrenamiento se llama `train.csv` y que la base de test `test.csv`, y que ambas siguen el formato establecido por la competencia. Para facilitar la experimentación, el archivo de entrada con la descripción del experimento tendrá la siguiente estructura:

- La primera línea contendrá el path a la(s) base(s) de datos, k, α, γ y K.
- Luego, habra K líneas de 42.000 valores, uno por cada muestra de la base de entrenamiento, donde un 1 indicará que esa muestra se considera parte del entrenamiento, y un 0 que se considera parte del test. Luego, de esta forma se pueden codificar las particiones realizadas por el *K-fold cross validation*.

El archivo de salida obligatorio tendrá para cada partición que figure en el archivo de entrada un vector con los α autovalores de la matriz de covarianza de mayor magnitud, con una componente del mismo por línea. Seguido a este primer vector debe haber un vector con los γ autovalores de las primeras γ iteraciones de la transformación PLS-DA, nuevamente con una componente por línea. Además, el programa deberá generar un archivo, cuyo formato queda a criterio del grupo, indicando la tasa de reconocimiento obtenida para cada partición. Adicionalmente, se generará un archivo que concatene la extension `.csv` al segundo valor recibido como parámetro del programa, que escribirá la predicción realizada sobre la base de test en el formato requerido por la competencia siguiendo el formato establecido por las reglas de la competencia.

Junto con el presente enunciado, se adjunta una serie de scripts hechos en `python` y un conjunto instancias de test que deberán ser utilizados para la compilación y un testeo básico de la implementación. Se recomienda leer el archivo `README.txt` con el detalle sobre su utilización.

[2] Notar que en la base original las figuras están rotadas y es blanco sobre negro, y no al revés.

Sobre la entrega

- Formato electrónico: Jueves 13 de Octubre de 2016, hasta las 23:59 hs., enviando el trabajo (`informe` + `código`) a `metnum.lab@gmail.com`. El `asunto` del email debe comenzar con el texto `[TP2]` seguido de la lista de apellidos de los integrantes del grupo. Ejemplo: `[TP2] Asad, Flores, Pompei`
- Formato físico: Viernes 14 de Octubre de 2016, de 17:30 a 18:00 hs.

5.2. Código fuente

5.2.1. Knn

```
int kNN(int k, vector<double>& img, vector< imgInfo >& bdd ){
        /**********CREO COLA DE PRIORIDAD**********/
        priority_queue < imgInfoDist > colaDePrioridad;

        /**********INSERTO LOS K DE MENOR DISTANCIA**********/
        for(int j=0 ; j < bdd.size(); j++){
                double dist = distancia(bdd[j]._imagen, img);
                imgInfoDist infoImagen;
                infoImagen._imagen = bdd[j]._imagen;
                infoImagen._etiqueta = bdd[j]._etiqueta;
                infoImagen._distancia = dist;
                colaDePrioridad.push(infoImagen);
                if(j > k-1 ){
                        colaDePrioridad.pop();
                }
        }

        /**********GENERO VECTOR DE ETIQUETAS**********/
        vector <int> etiquetas;
        for(int l = 0; l < k; l++){
                etiquetas.push_back(colaDePrioridad.top()._etiqueta);
                colaDePrioridad.pop();
        }

        /**********APLICO MODA**********/
        int resultado = moda(etiquetas);
        return resultado;
}
```

5.2.2. Knn2

Se adjunta el código de Knn Versión 2 o Knn Posicional.

```
double normaDos(vector<double>& x){
        double acum;
                for(int i=0; i < x.size(); i++){
                acum = pow((x[i]), 2);
        }
        return sqrt(acum);
}
```

```
int modaPosicional(vector <imgInfoDist>& imgs, vector<double>& original
    ){
        double normOrig = normaDos(original);
        double zeroFive = normOrig * 0.05;
        double oneFive = normOrig * 0.15;
        double twoFive = normOrig * 0.25;
        double fourZero = normOrig * 0.40;
        vector<int> posiciones(10);
        for(int k=0; k < 10 ; k++){
                posiciones[k] = 0;
        }
```

```
        for(int i=0; i < imgs.size() ; i++){
                if(imgs[i]._distancia <= zeroFive){
                        posiciones[imgs[i]._etiqueta] += 12;
                }
                else if(imgs[i]._distancia <= oneFive){
                        posiciones[imgs[i]._etiqueta] += 7;
                }
                else if(imgs[i]._distancia <= twoFive){
                        posiciones[imgs[i]._etiqueta] += 4;
                }
                else if(imgs[i]._distancia <= fourZero){
                        posiciones[imgs[i]._etiqueta] += 2;
                }
                else{
                        posiciones[imgs[i]._etiqueta] += 1;
                }
        }
        int actual = 0;
        int moda = 0;
        for(int j=0; j < 10 ; j++){
                if(actual < posiciones[j]){
                        actual = posiciones[j];
                        moda = j;
                }
        }
        return moda;
}
```

```
int kNNVersion2(int k, vector<double>& img, vector< imgInfo >& bdd ){
        /**********CREO COLA DE PRIORIDAD**********/
        priority_queue < imgInfoDist > colaDePrioridad;

        /**********INSERTO LOS K DE MENOR DISTANCIA**********/
        for(int j=0 ; j < bdd.size(); j++){
                double dist = distancia(bdd[j]._imagen, img);
                imgInfoDist infoImagen;
                infoImagen._imagen = bdd[j]._imagen;
                infoImagen._etiqueta = bdd[j]._etiqueta;
                infoImagen._distancia = dist;
                colaDePrioridad.push(infoImagen);
                if(j > k-1 ){
                        colaDePrioridad.pop();
                                        // Mantengo k elementos
                            dentro de cola
                }
        }

        /**********GENERO VECTOR DE ETIQUETAS**********/
        vector <imgInfoDist> etiquetas;
        for(int l = 0; l < k; l++){
                etiquetas.push_back(colaDePrioridad.top());
                colaDePrioridad.pop();
        }

        /**********APLICO MODA**********/
        int resultado = modaPosicional(etiquetas, img);
```

```
        return resultado;
}
```