Métodos de cambio de base sobre sistemas de ecuaciones y sus aplicaciones en la categorización de imágenes

3 de agosto de 2016 — Métodos Numéricos

Integrante	LU	Correo electrónico
Arribas, Joaquín Manuel	702/13	joacoarribas@hotmail.com
Lebrero Rial, Ignacio Manuel	751/13	ignaciolebrero@gmail.com
Vázquez, Jésica	318/13	jesis_93@hotmail.com

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

Resumen

Los sistemas de ecuaciones lineales se utilizan para resolver y modelar una gran variedad de problemas en diferentes áreas, tales como física, ingeniería y ciencias sociales. En el presente Trabajo Práctico se aborda la problemática de determinar el dígito que contiene una imagen en base a otras imagenes que sí sabemos que dígito contienen. Para ello utilizamos los métodos denominados PCA y PSDLA los cuales reformularán el problema como la resolución de un sistema de ecuaciones lineales. En base a los resultados de estos métodos se utilizara el algoritmo KNN para determinar en última instancia el dígito de una determinada imagen.

Palabras clave: método, PCA, PLS, KNN, autovalores, autovectores

Índice

1. Introducción

El presente trabajo práctico consiste en, dado un conjunto de imágenes que representan dígitos manuscritos del 0 al 9, *entrenar* un método de clasificación que permita reconocerlos. El objetivo será, dada una nueva imagen, estimar a que dígito corresponde.

Las imágenes de los dígitos manuscritos se representan como un vector $x_i \in \mathbb{R}^m$ donde cada posición corresponde a un píxel. Sabemos que el tamaño de las imágenes es de 28x28, por lo cual el tamaño del vector será 784. Entonces podemos representar al conjunto de imágenes como una matriz $A \in \mathbb{R}^{nxm}$ donde n corresponde a la cantidad de imágenes, y m al tamaño de cada una. Por cada fila de la matriz A tendremos los píxeles correspondientes a una imagen.

$$A = \begin{bmatrix} Imagen_1 \\ Imagen_2 \\ \vdots \\ Imagen_n \end{bmatrix}$$

1.1. k-Nearest neighbors

El método kNN (k vecinos más cercanos) es un método de clasificación *supervisada*, donde por supervisado entendemos que la estimación realizada se hace en base a un conjunto de datos de entrenamiento (las imágenes que sabemos a qué dígito corresponden). El método calcula la probabilidad de que un nuevo elemento pertenezca a una determinada clase o conjunto de los datos ya procesados. Para determinar dicha pertenencia se consideran los k *vecinos* más cercanos.

1.2. Principal Component Analysis

El método PCA (análisis de componentes principales) es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos, conservando las características importantes de estos. Se busca realizar una transformación de los datos que disminuya la redundancia (la covarianza entre los datos). En nuestro caso reduciremos el tamaño de las imágenes, quedándonos con la fracción que capture la mayor varianza entre las ellas. Luego, al recibir una nueva imagen, se le realiza la misma transformación y se busca, en nuestro caso mediante kNN, a que dígito corresponde.

1.3. Partial Least Squares Regression

El método PLS (regresión de mínimos cuadrados parciales) es una técnica de la misma índole que PCA dado que inicialmente busca realizar una transformación sobre los datos para conseguir las mismas propiedades que en PCA. La mayor diferencia entre este método y el anterior es que este es un método *supervisado*, utiliza la información obtenida durante el proceso de entrenamiento del clasificador. En cuanto a lo realizado por el método, difieren en cómo realizar la transformación del espacio para reducir la dimensionalidad de los datos.

2. Estructura

La estructura utilizada para representar la matriz es un vector de vectores donde por cada fila se tiene una imagen. Además se utilizan otros dos vectores llamados *etiqueta* y *estimación*. En el primero se almacena la etiqueta correspondiente a cada imagen de la matriz y el segundo se utiliza para almacenar su estimación, es decir, a qué clase de dígito pertenece (una vez realizado el método kNN).

3. Desarrollo

3.1. kNN

El método kNN es un proceso mediante el cual se pueden clasificar datos. Dado un conjunto de datos donde cada uno pertenece a un grupo característico, se intenta estimar para cada nuevo dato obtenido a qué conjunto pertenece. El conjunto del cual para cada dato se conoce su clase característica se denomina conjunto de *entrenamiento*.

Para poder estimar a qué clase pertenece un nuevo dato se le calcula la distancia a cada dato de la base de entrenamiento. Luego se buscan los k elementos que minimizan dicha distancia y se determina cuál es el conjunto que predomina sobre esos datos. Ese conjunto será la estimación del nuevo dato.

3.1.1. El algoritmo

En esta sección analizaremos la complejidad temporal del algoritmo y exhibiremos su pseudocódigo.

Como dijimos previamente, el algoritmo consiste en calcular para cada imagen a la cual se quiere estimar su clase característica, la distancia a cada uno de los elementos del conjunto de entrenamiento.

El algoritmo recibe como *input* el conjunto de imágenes utilizadas como base de entrenamiento del categorizador, otro que tiene imágenes sin etiqueta, y la cantidad de vecinos a contemplar. Luego, para cada imagen que queremos estimar a qué clase de dígito manuscrito pertenece le calculamos su *distancia* a las todas las que están como base de entrenamiento, y se eligen los k más cercanos (se busca minimizar la norma dos de la resta de las dos imágenes). De esos k vecinos se calcula la clase característica que más *representa* a ese conjunto de imágenes (el dígito que más aparece entre los k vecinos). Para cada imagen que se quiere estimar su grupo característico se le asigna ese dígito como estimación de etiqueta.

El pseudocódigo del algoritmo es el siguiente:

Pseudocodigo 1 **Matriz** kNN(**Matriz** imágenesTrain, **Matriz** imagenesTest, **int** vecinos) $\mathcal{O}(n^2 * (m + vecinos)))$

for cada imagen en *imagenesTest* **do**	$\mathcal{O}(n^2 * m)$
Estimarle su etiqueta respecto de las imágenes en *imagenesTrain*	$\mathcal{O}(n * m + vecinos * n)$
Asignarle la estimación como etiqueta	$\mathcal{O}(1)$
end for	

El pseudocódigo del algoritmo para estimar la etiqueta es el siguiente:

Pseudocodigo 2 **int** dameEstimación(**Matriz** imágenesTrain, **vector(double)** imagenAEstimar, **int** vecinos) $\mathcal{O}(n * m + vecinos * n)$

Sea x un vector de tamaño igual que *imagenAEstimar*	$\mathcal{O}(m)$
Sea y un vector de tamaño igual a la cantidad de imágenes en *imágenesTrain*	$\mathcal{O}(n)$
for cada imagen en *imágenesTrain* **do**	$\mathcal{O}(n * m)$
Guardar en x la resta (distancia) entre *imagenAEstimar* y la imagen de *imágenesTrain*	$\mathcal{O}(m)$
Guardar en y la norma dos del vector x	$\mathcal{O}(m)$
end for	
Encontrar las k imágenes que minimizan la norma 2	$\mathcal{O}(vecinos * n)$
Elegir como etiqueta la más utilizada por las k imágenes	$\mathcal{O}(1)$

En los anteriores pseudocódigos n es la cantidad de filas de la matriz de entrada y m la cantidad de columnas. La cantidad de vecinos está acotada por la cantidad de filas de la matriz.

3.2. Método de la Potencia

3.2.1. Motivación

El método de la potencia es un algoritmo utilizado para calcular el autovector y autovalor dominante de una matriz. Introducimos este método dado que es utilizado por los métodos subsiguientes. Las características que tiene que cumplir una matriz para poder calcularle sus autovectores y autovalores correspondientes mediante este método son las siguientes. Sea $A \in \mathbb{R}^{nxn}$:

- Sean $\lambda_1, ..., \lambda_n$ autovalores de A, se tiene que cumplir que $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n| > 0$.
- Sean $\{v_1, ..., v_n\}$ los autovectores asociados a los autovalores de A, el conjunto de autovectores debe formar una *base*.

La idea es calcular en cada iteración el mayor autovalor de la matriz A.

3.2.2. El Algoritmo

En esta sección analizaremos la complejidad temporal del algoritmo y exhibiremos su pseudocódigo.

Pseudocodigo 3 **Matriz** Método_Potencia(**Matriz** A $\in \mathbb{R}^{nxn}$, **Vector** x $\in \mathbb{R}^n$)	O(kn^2)
Definir Vector $y \in \mathbb{R}^n$	$\mathcal{O}(n)$
$y =$ Ax	$\mathcal{O}(n^2)$
while no se cumple la condición de convergencia **do**	
Normalizar y	$\mathcal{O}(n)$
$x = y$	$\mathcal{O}(n)$
$y =$ Ax //en y queda el autovector	$\mathcal{O}(n^2)$
end while	
Devolver máximo valor absoluto de de y	$\mathcal{O}(n)$

Donde la condición de convergencia puede estar dada, en nuestro caso, por la cantidad de iteraciones que realiza el **while** o por $\|$y$\|_\infty$ - $\|$x$\|_\infty < \epsilon$.

Para poder analizar la complejidad temporal del algoritmo, se requiere que el vector y converja en menos de k iteraciones, con lo cual la complejidad final del algoritmo es $O(kn^2)$.

3.3. PCA

El método PCA es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos, conservando las características importantes de estos. Se busca realizar una transformación de los datos que disminuya la redundancia entre ellos.

El proceso es el siguiente:

Sean $x^{(1)}, ..., x^{(n)}$ las n muestras de datos, en nuestro caso las imágenes con $x^{(i)} \in \mathbb{R}^m$ (cada componente del vector representa un píxel) , e $Imágenes \in \mathbb{R}^{nxm}$ la matriz que contiene en cada fila una imagen (i.e. $fila_i(Imágenes) = (x^{(i)})^t$).

- Se calcula la media entre todas las imágenes. Esta se calcula componente a componente (i.e. $\mu = \frac{(x^{(1)}+...+x^{(n)})}{n}$).
- Se calcula la matriz de covarianza. Para esto, dadas n observaciones de 2 variables x_k, x_j la covarianza entre éstas se obtiene de la siguiente manera:

$$\sigma_{x_j x_k} = \frac{1}{n-1} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k) \tag{1}$$

  Sea $v = (1, ..., 1)^t$, puedo reescribir la covarianza como:

$$\sigma_{x_j x_k} = \frac{1}{n-1}(x_k - \mu_k v)^t (x_j - \mu_j v) = \frac{1}{\sqrt{n-1}}(x_k - \mu_k v)^t \frac{1}{\sqrt{n-1}}(x_j - \mu_j v) \tag{2}$$

  Siendo x_k y x_j vectores correspondientes los píxeles k y j (respectivamente) de todas las muestras, μ_k y μ_j la esperanza.

  Defino a la matriz $\mathrm{X} \in \mathbb{R}^{nxm}$ como:

$$X = \begin{bmatrix} \frac{x_1^{(1)}-\mu_1}{\sqrt{n-1}} & \frac{x_2^{(1)}-\mu_2}{\sqrt{n-1}} & \cdots & \frac{x_m^{(1)}-\mu_m}{\sqrt{n-1}} \\ \frac{x_1^{(2)}-\mu_1}{\sqrt{n-1}} & \frac{x_2^{(2)}-\mu_2}{\sqrt{n-1}} & \cdots & \frac{x_m^{(2)}-\mu_m}{\sqrt{n-1}} \\ \vdots & \vdots & & \vdots \\ \frac{x_1^{(n)}-\mu_1}{\sqrt{n-1}} & \frac{x_2^{(n)}-\mu_2}{\sqrt{n-1}} & \cdots & \frac{x_m^{(n)}-\mu_m}{\sqrt{n-1}} \end{bmatrix} \tag{3}$$

  Con lo cual, la matriz de covarianza es $\mathrm{M}_X = \mathrm{X}^t\mathrm{X}$.

- Se calcula una base ortonormal de autovectores de M_X. Esta base es calculable mediante el método de la potencia dado que la matriz de covarianza M_X cumple los requisitos necesarios explicados en el método.
- Se utilizan sólo los primeros α autovectores de dicha base.
- Se multiplica a la matriz de imágenes por los primeros α autovectores. De esta manera se realiza la transformación característica de la matriz (el cambio de base).

Por último, una vez obtenida la transformación característica del método, para cada nueva imagen que se quiera estimar su *etiqueta*, sólo es necesario aplicarle la misma transformación característica (multiplicarla por los α autovectores).

3.3.1. El Algoritmo

En esta sección analizaremos la complejidad temporal del algoritmo y exhibiremos su pseudocódigo.

El algoritmo recibe como *input* el conjunto de imágenes utilizadas como base de entrenamiento del categorizador, y retorna los primeros α autovectores para poder aplicarle a cada subsiguiente imagen la transformación característica.

El pseudocódigo del algoritmo es el siguiente:

Pseudocodigo 4 **Matriz** PCA(**Matriz** imágenesTrain $\in \mathbb{R}^{nxm}$, **int** α)	$\mathcal{O}(\alpha m(mk+n)$
Calcular $\mu = \frac{(x^{(1)}+...+x^{(n)})}{n}$	$\mathcal{O}(nm)$
Calcular matriz X como en la ecuación (3)	$\mathcal{O}(nm)$
Definir M_X como X^tX	$\mathcal{O}(m^2n)$
$V \in \mathbb{R}^{mx\alpha}$ = Calcular_Matriz_Ortornormal(M_X, α)	$\mathcal{O}(\alpha km^2)$
Aplicar a **imágenesTrain** la transformación característica (multiplicarla por V)	$\mathcal{O}(\alpha nm)$
Devolver V	$\mathcal{O}(1)$

En el pseudocódigo n es la cantidad de filas de *imágenesTrain* y m la cantidad de columnas.

La complejidad final del algoritmo es $O(\alpha m(mk+n))$ siendo k la cantidad máxima de iteraciones permitidas para el método de la potencia.

El pseudocódigo que calcula la matriz ortonormal es el siguiente:

Pseudocodigo 5 **Matriz** Calcular_Matriz_Ortonormal(**Matriz** $M_X \in \mathbb{R}^{mxm}$, **int** α)	$\mathcal{O}(\alpha km^2)$
Definir Matriz Res $\in \mathbb{R}^{mx\alpha}$	$\mathcal{O}(m\alpha)$
for i desde 1 a α **do**	$\mathcal{O}(\alpha)$
Definir vector v_i de dimensión m con valores aleatorios	$\mathcal{O}(m)$
λ_i = método_potencia(M_X,v_i) //v_i es el autovector asociado al autovalor λ_i	$\mathcal{O}(km^2)$
normalizar v_i	$\mathcal{O}(m)$
Aplicar deflación a M_X con v_i	
Columna$_i$(Res) = v_i	$\mathcal{O}(m)$
end for	
Devolver Res	$\mathcal{O}(1)$

La complejidad final del algoritmo es $O(km^2)$, siendo k la cantidad máxima de iteraciones permitidas para el método de la potencia.

Para aplicar deflación:

Actualizo la matriz M_X tal que $M_X^{(i+1)} = M_X^{(i)} - \lambda_i v_i v_i^t$. La matriz $M_X^{(i+1)}$ tiene autovalores 0,...,0,λ_{i+1},...,λ_n.

3.4. PLSDA

Si bien PCA obtiene una transformación para reorganizar el espacio que estamos usando, lo hace de manera *no supervisada*. El método PLSDA utiliza fundamentos parecidos a PCA pero utiliza los valores de las clases a las que los datos pertenecen para obtener más información a la hora de armar la transformación.

Sea $X \in \mathbb{R}^{nxm}$ la matriz definida en PCA. Sean $c_1, ..., c_n$ tal que c_i es la clase a la que pertence la entrada x_i de X y c_{cant} la cantidad de clases distintas que pertenecen a X.
Definimos $preY \in \mathbb{R}^{nxc_{cant}}$ tal que:

$$preY_{ij} = \begin{cases} 1 & si \quad j = c_i \\ -1 & si \quad \text{caso contrario} \end{cases} \tag{4}$$

De esta manera la columna i de $preY$ nos determina para toda muestra x_j si pertenece a la clase i.

A continuación, sea $\mu_1, ..., \mu_{c_{cant}}$ el promedio de cada columna de $preY$. Definimos Y como la matriz que por cada columna tiene la columna normalizada de $preY$.

$$Y_{ij} = (preY_{ij} - \mu_j)/\sqrt{n-1} \tag{5}$$

Luego, de la misma manera que en el método anterior, se calcula la matriz de covarianza M_X. Como en este caso el método es supervisado, se utiliza la matriz Y para calcularla:

$$M_X = X^t * Y * Y^t * X \tag{6}$$

Luego se le calcula la base de autovectores a la matriz de covarianza y se le aplica la transformación característica al conjunto de imágenes inicial. De esta manera se le reduce el tamaño a las imágenes, dejándole solo γ componentes. Esta base es calculable mediante el método de la potencia dado que la matriz de covarianza M_X cumple los requisitos necesarios explicados en el método.

Por último, una vez obtenida la transformación característica del método, para cada nueva imagen que se quiera estimar su *etiqueta*, sólo es necesario aplicarle la misma transformación característica (multiplicarla por los γ autovectores).

3.4.1. El algoritmo

En esta sección analizaremos la complejidad temporal del algoritmo y exhibiremos su pseudocódigo.

El algoritmo recibe como *input* el conjunto de imágenes utilizadas como base de entrenamiento del categorizador, y retorna los primeros γ autovectores para poder aplicarle a cada imagen subsiguiente la transformación característica.

El pseudocódigo del algoritmo es el siguiente:

Pseudocodigo 6 **Matriz** PLSDA(**Matriz** imágenesTrain, **int** γ)	$\mathcal{O}(m^2 * n)$
Calcular $\mu = \frac{(x^{(1)}+...+x^{(n)})}{n}$	$\mathcal{O}(n * m)$
Calcular matriz X como en la ecuación (3)	$\mathcal{O}(n * m)$
Calcular matriz preY como en la ecuación (4)	$\mathcal{O}(n)$
Calcular matriz Y como en la ecuación (5)	$\mathcal{O}(n)$
Definir M_X como en la ecuación (6)	$\mathcal{O}(n * m)$
Calcular $V \in \mathbb{R}^{m*\alpha}$ los primeros α autovectores de la base ortonormal de M_A	$\mathcal{O}(\gamma * k * m^2)$
Aplicar a **imágenesTrain** la transformación característica (multiplicarla por V)	$\mathcal{O}(n * m * \alpha)$
Retornar V	$\mathcal{O}(1)$

En el pseudocódigo n es la cantidad de filas de *imágenesTrain* y m la cantidad de columnas.

Para calcular la base de autovectores es necesario disminuir en cada iteración el tamaño de las matrices X e Y. Luego de cada iteración (luego de calcular un autovector) se actualizan las matrices X e Y de la siguiente manera:

- Se calcula el vector $t_i \in \mathbb{R}^m$ como $X * autovector_i$.
- Se genera una matriz $T \in \mathbb{R}^{m*m}$:

$$T = ti * ti^t \tag{7}$$

- Se actualiza la matriz X:

$$X = X - T * X \tag{8}$$

- Se actualiza la matriz Y:

$$Y = Y - T * Y \tag{9}$$

El pseudocódigo para calcular la base de autovectores es el siguiente:

Pseudocodigo 7 **Matriz** CalcularMatrizOrtonormal(**Matriz** M_X, **int** γ)	$\mathcal{O}(m^2 * n)$
Definir vector **v** de dimensión m con valores aleatorios	$\mathcal{O}(m)$
for i desde 1 a γ **do**	$\mathcal{O}(k * \gamma * m^2)$
Obtener v_i autovector con el método de la potencia	$\mathcal{O}(k * m^2)$
Se calcula $t_i = X * v_i$	$\mathcal{O}(m^2)$
Se normaliza t_i	$\mathcal{O}(m^2)$
Se calcula la matriz T como en la ecuación (7)	$\mathcal{O}(m^2)$
Se actualiza la matriz X como en la ecuación (8)	$\mathcal{O}(m^2)$
Se actualiza la matriz Y como en la ecuación (9)	$\mathcal{O}(m^2)$
end for	

3.5. Análisis estadístico

Como mencionamos en la introducción el objetivo de este trabajo es, dado un conjunto de imágenes que representan dígitos manuscritos, *entrenar* un clasificador que permita reconocerlos. Para eso, contamos con una base de datos sobre la cual sabemos, para cada imagen, que dígito le corresponde. A la hora de realizar un análisis estadístico sobre los resultados obtenidos y representar así la efectividad de los métodos, utilizamos las métricas explicadas a continuación. Previamente definiremos 3 conceptos. Dada una clase i de datos:

- **Verdaderos positivos**: Son las muestras que pertenecían a la clase i y fueron efectivamente identificadas como tal. Lo notaremos como t_{p_i}
- **Falsos positivos**: Son las muestras que fueron identificadas como pertenecientes a la clase i, cuando en realidad no lo eran. Lo notaremos como f_{p_i}
- **Falsos negativos**: Son las muestras que pertenecían a la clase i, pero fueron identificadas con otra clase. Lo notaremos como f_{n_i}

Las métricas utilizadas para el análisis estadístico son:

- **Hit rate**: Representa la cantidad de dígitos correctamente clasificados respecto de la cantidad total de dígitos.
- **Precisión**: Es una medida que representa la cantidad de aciertos relativos que tiene un clasificador respecto de una clase particular. La precisión se calcula como: $\frac{t_{p_i}}{t_{p_i}+f_{p_i}}$
- **Recall**: Es una medida que representa la cantidad de veces que el clasificador identifica correctamente a las muestras pertenecientes a la clase i. El recall se calcula como: $\frac{t_{p_i}}{t_{p_i}+f_{n_i}}$

3.6. k-Fold Cross Validation

El método *k-Fold Cross Validation* es una técnica utilizada para evaluar los resultados de un análisis estadístico y poder garantizar que son independientes de la partición de datos de entrenamiento y prueba. El particionamiento de los datos entre entrenamiento y prueba se realiza sobre la base de todos los datos de los cuales se sabe su clase. El método funciona de la siguiente manera:

1. Se desordenan los datos.
2. Se separan los datos en k folds (particiones) del mismo tamaño.
3. Para $i = 1...k$:
   Entrenar sobre todos los folds menos el i-ésimo, y validar (estimar) sobre ese.

Una vez obtenidos los resultados de cada iteración, se realiza el promedio de todos. De esta manera podemos hacer un análisis más riguroso respecto de los resultados ya que nos aseguramos no utilizar los métodos siempre con el mismo conjunto de datos de entrenamiento y prueba.

Los valores que utilizaremos para K en los experimentos serán 5, 10 y 15. La razón por la cual elegimos estos valores es que consideramos que es un rango adecuado para experimentar. La base de datos utilizada para los experimentos que exhibimos a continuación consistió en 42000 imágenes de las cuales sí se sabe, para cada una, a qué clase pertenece.

Utilizando un K mayor que 15 se estaría utilizando una gran porción de los datos para el entrenamiento del clasificador, lo cual podría producir un *sobreentrenamiento*, haciendo que tenga más información que la necesaria a la hora de clasificar una nueva imagen, y como consecuente, clasificándola de manera errónea. Con K igual a 15 se entrena al clasificador con 39200 imágenes, un 93.3 % del total de los datos, y por ende se prueba con 2800 imágenes de las cuales se supone que no se sabe la etiqueta.

Utilizando un K menor que 5 nos iríamos al otro extremo y estaríamos utilizando muchos datos para testear el método, sin quizás haberlo entrenado tanto previamente. Con K igual a 15 se entrena al clasificador con 33600 imágenes, un 80 % de los datos, y por ende se prueba con 8400 imágenes de las cuales se supone que no se sabe la etiqueta.

4. Experimentación

En esta sección analizaremos los experimentos que realizamos para el análisis cuantitativo y cualitativo, respecto de los métodos y algoritmos implementados para el trabajo práctico.

4.1. Primer Experimento

Como mencionamos en la sección de desarrollo, la idea del algoritmo de clasificación kNN es estimar la pertenencia de un objeto a una clase. Sin embargo, este método puede funcionar de diversas maneras dependiendo de la cantidad de vecinos a contemplar. En el siguiente experimento nos proponemos analizar, fijando la cantidad de dimensiones del método PLSDA y la cantidad de *folds* del *k-fold cross validation* en 10, variando la cantidad de vecinos utilizadas por el método kNN en el rango [10, 100] aumentando de a 10 en cada iteración, los siguientes items:

- ¿Cuánto y cómo varía el *hit rate*, la *precisón* y el *recall* del método al aumentar la cantidad de vecinos? No creemos que al aumentar la cantidad de vecinos a tener en cuenta refleje necesariamente un impacto positivo en las mediciones estadísticas del método. Puede suceder que, al aumentar la cantidad de vecinos, se contemple un espacio mayor del que se necesita para que el método resulte eficiente. Por ejemplo:

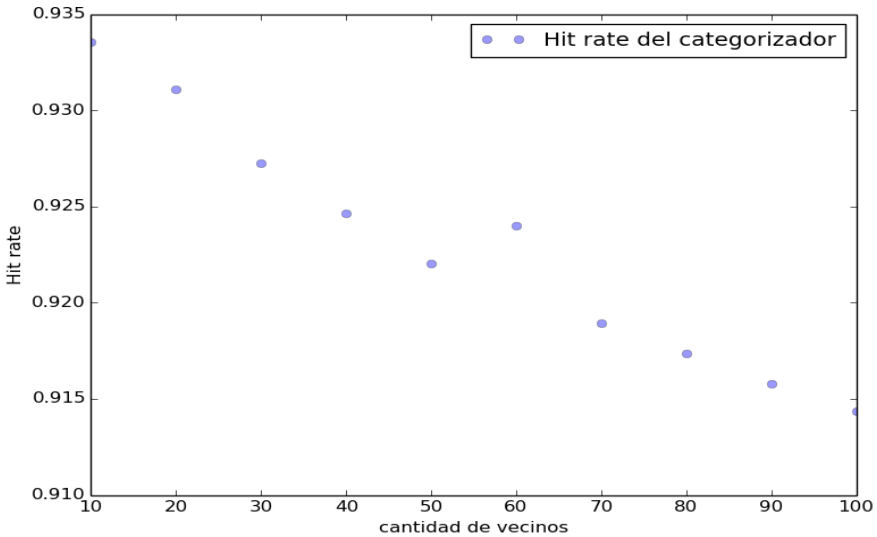
  Sea x una imagen que representa un dígito cero y todavía no fue clasificada. Por distintas cuestiones, x se parece mucho a otras 10 imágenes que ya fueron clasificadas como cero, y luego, sus próximas 20 imágenes más cercanas son ochos. Si se utiliza kNN con sólo 10 vecinos el método clasificaría bien a x. Sin embargo, si la cantidad de vecinos fuese 30, el método clasificaría a x como un ocho, cuando en realidad es un cero.

- ¿Como varía el tiempo que tarda el método? Como explicamos en la sección de desarrollo, el método tiene una complejidad que aumenta linealmente en relación a la cantidad de vecinos. Creemos que esta relación se cumplirá.

Lo que tratamos de encontrar mediante este experimento es qué cantidad de vecinos optimizaría el método (categorizaría mejor las imágenes) para una cantidad razonable de dimensiones.

Los resultados fueron los siguiente:

- Las mediciones respecto del *hit rate* fueron las siguientes:

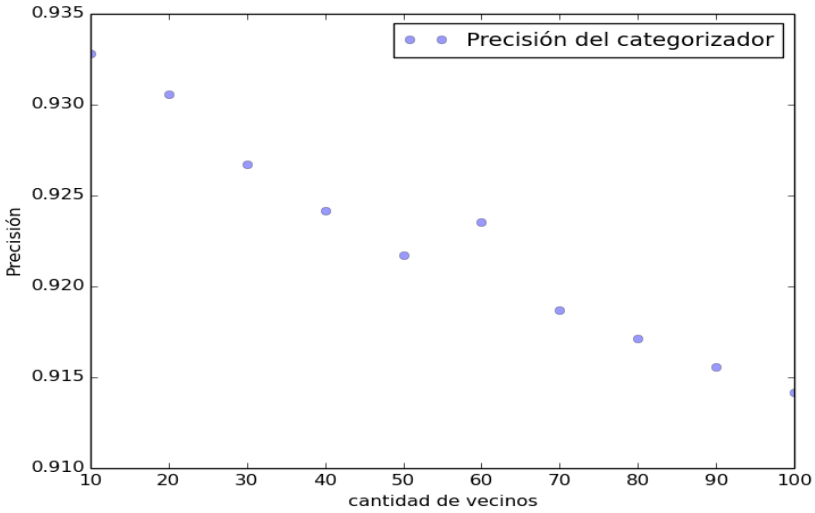


  Este gráfico representa, por cada iteración del *K-fold cross validation*, el promedio de los *hit rates* por cada iteración de vecinos distinta. Se puede ver como a medida que aumenta la cantidad de vecinos disminuye el *hit rate*. Podemos observar como nuestra hipótesis se validó para esta cantidad de

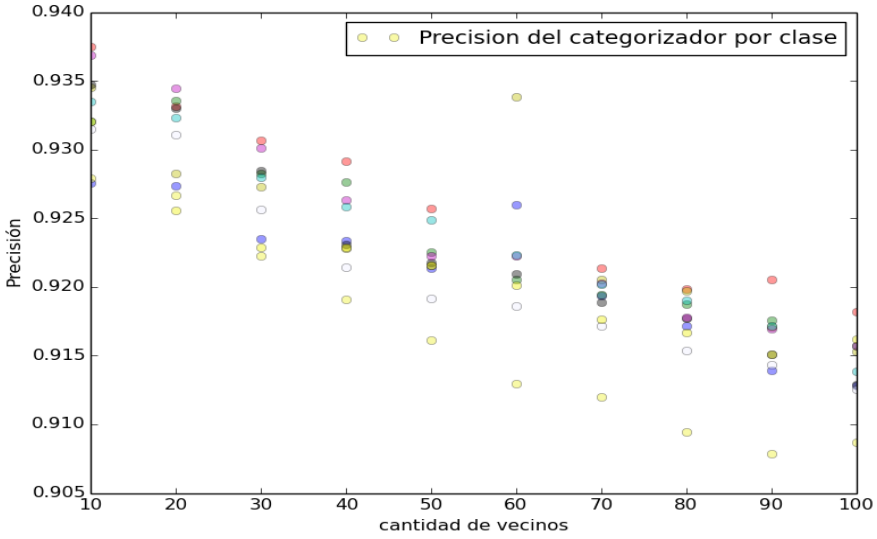
dimensiones y efectivamente aumentar la cantidad de vecinos no refleja necesariamente un impacto positivo en el método. La razón detrás de esto creemos que es el ejemplo que mencionamos antes.

En el gráfico se puede notar como hay una variación *anormal* cuando la cantidad de vecinos es 60. Para analizar dicha anomalía observamos que sucedía con la precisión.

- Las mediciones respecto de las *precisiones* fueron las siguientes:



Como se puede observar, la medición de *precisión* está estrechamente relacionada con el *hit rate* (se comportan de manera análoga). También notamos una anomalía en este gráfico cuando la cantidad de vecinos a considerar es 60. Para obtener un poco más de información al respecto de qué estaba ocurriendo nos fijamos en el promedio de las precisiones de cada clase (etiqueta). Los resultados fueron los siguientes:

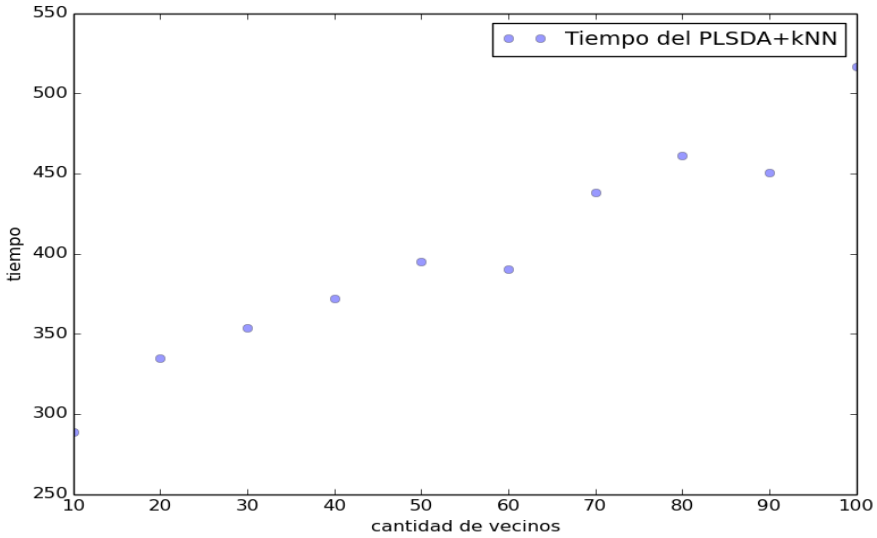


Notamos que por alguna razón la precisión de la clase *cero* era muy bien estimada, es decir, muy pocas veces sucedió que una una imagen se estime como cero cuando no lo era, al considerar 60 vecinos en el método kNN. Consideramos que esto sucede porque, para cada imagen que se quiere estimar y es un cero:

- Al considerar vecinos en el rango [10, 50] la cantidad de *ceros* cercanos a las imágenes a estimar deben estar casi igualadas por una cantidad de imágenes que tiene otra etiqueta

- Al considerar 60 vecinos deben entrar en el rango de cercanía muchos ceros, por lo cual se estimará bien a todas las imágenes que lo sean (y se reducirán a su vez los *falsos positivos*).
- Una vez pasado este rango se vuelve a equiparar con la cercanía a otras imágenes de otras etiquetas.

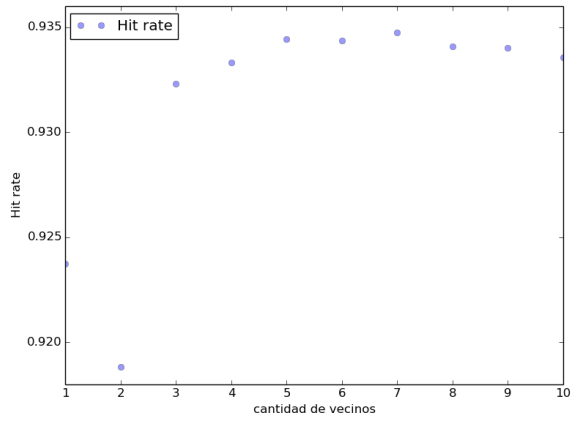
- Las mediciones respecto del *tiempo* fueron las siguientes:

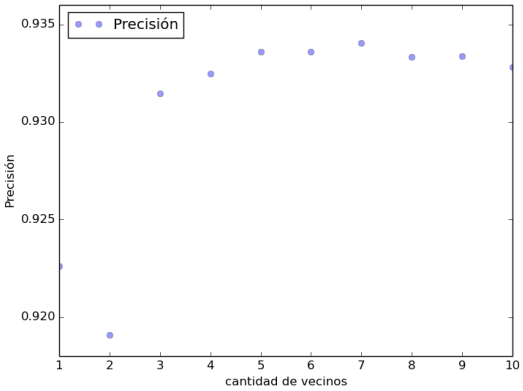


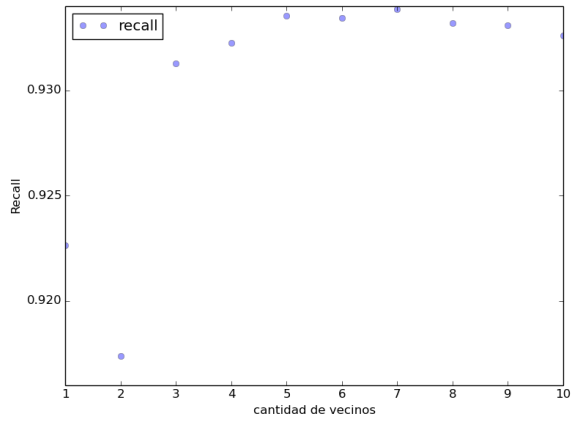
  Efectivamente, como mencionamos en la sección de desarrollo, el tiempo que tarda el método varía linealmente respecto de la cantidad de vecinos a contemplar en kNN. Por ende no solo no mejora en términos de eficiencia, sino que también requiere mayor tiempo de cómputo.

Con estas mediciones podemos concluir que el rango en el cual se encuentra la cantidad de vecinos que mejora notablemente la eficiencia del método es alrededor del 10. Como originalmente la intención de este experimento fue encontrar el valor que *optimice* el método, volvimos a correr los experimentos, pero esta vez variando la cantidad de vecinos en el rango [1, 10] aumentando de a 1 (dado que el máximo había sido con vecinos igual a 10).

Los resultados fueron los siguientes:







Podemos observar como la cantidad de vecinos que nos maximiza el *beneficio* para esta cantidad de dimensiones es el 7. Por beneficio entendemos a que es el número que nos da en promedio:

- Mayor cantidad de aciertos (verdaderos positivos). Para cada imagen a estimar es el parámetro que mejor acierta qué etiqueta le corresponde.
- Menor cantidad de desaciertos (falsos positivos y falsos negativos).

Con 7 vecinos el método clasificó bien el 93,48 % de las imágenes test. En base a este valor trataremos de fijar las otras variables para llegar una combinación que optimice nuestro uso del método.

Otra observación que podemos hacer de los gráficos es que al tener en cuenta solo dos vecinos, el hitrate decrece alrededor de 0,015 puntos con respecto al mismo análisis pero con otra cantidad de vecinos. Creemos que esto se debe a que, al estar comparando sólo entre dos vecinos, el algoritmo estima de manera errónea la etiqueta. Al ser dos las clases más *cercanas* a la imagen a estimar, el algoritmo selecciona la clase de mayor valor númerico, generando así estos errores.

4.2. Segundo Experimento

Como mencionamos en la sección de desarrollo, la idea de los algoritmos de cambio de base es reducir la dimensión de las imágenes (a su vez que disminuyen el ruido que pueden proporcionar), además de disminuir su complejidad temporal. En el siguiente experimento nos proponemos analizar, fijando la cantidad de vecinos a tener en cuenta en el método kNN en 7 (valor obtenido en el anterior experimento), la cantidad de *folds* del *k-fold cross validation* en 10, variando las dimensiones del método $PLSDA$ en el rango [5, 60] aumentando de a 5 en cada iteración, los siguientes items:

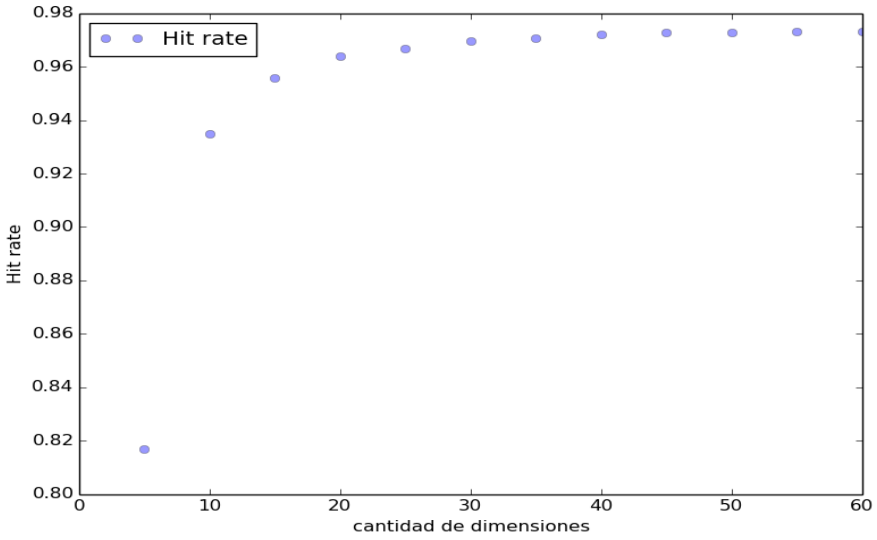
- ¿Cuánto y cómo varía el *hit rate* del método al aumentar la cantidad de dimensiones? En un principio creemos que debería ser más *eficiente* el método al aumentar la cantidad de dimensiones. Por eficiente nos referimos a que, al aportarle más información al método kNN para seleccionar mejor la etiqueta a la cual pertenece, el categorizador funcionará mejor.

- ¿Cuánto y cómo varía la *precisión* del método al aumentar la cantidad de dimensiones? Creemos que el categorizador debería estimar mejor la clase característica de cada nueva imagen dado que tiene más *información* para hacerlo, por lo cual la precisión debería ser mejor en cada iteración.

Lo que tratamos de encontrar mediante este experimento es qué cantidad de dimensiones optimizaría el método (categorizaría mejor las imágenes) para la cantidad de vecinos ya determinada en el anterior experimento.

Los resultados fueron los siguientes:

- Las mediciones respecto del *hit rate* fueron las siguientes:

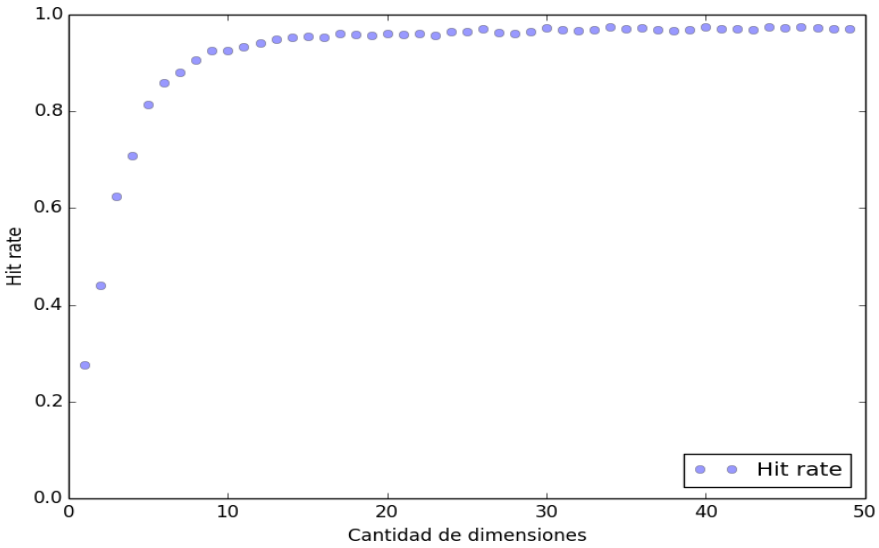


Este gráfico representa, por cada iteración del *K-cross-validation*, el promedio de los *hit rates* por cada iteración de dimensión distinta. Se puede ver como a medida que aumenta la cantidad de dimensiones aumenta el *hit rate*, llegando al máximo en 55 y decreciendo levemente en 60. Podemos observar entonces cómo nuestra hipótesis se validó. El porcentaje de imágenes que fueron bien estimadas utilizando 55 dimensiones fue un 97,329 %.

Es remarcable notar la diferencia que hay entre usar 5 y 10 dimensiones. Al ir tomando los autovalores más grandes de la matriz de covarianza, las primeras iteraciones tienen un impacto muy grande en la performance, ya que por cada dimensión agregada se está tomando mucha información relevante. Suponemos que la razón por la cual la pendiente decrece rápidamente es que las imágenes tienen información relevante en lugares muy particulares, con lo cual la mayor cantidad de esta queda guardada en las dimensiones cuyos autovalores son mayores.

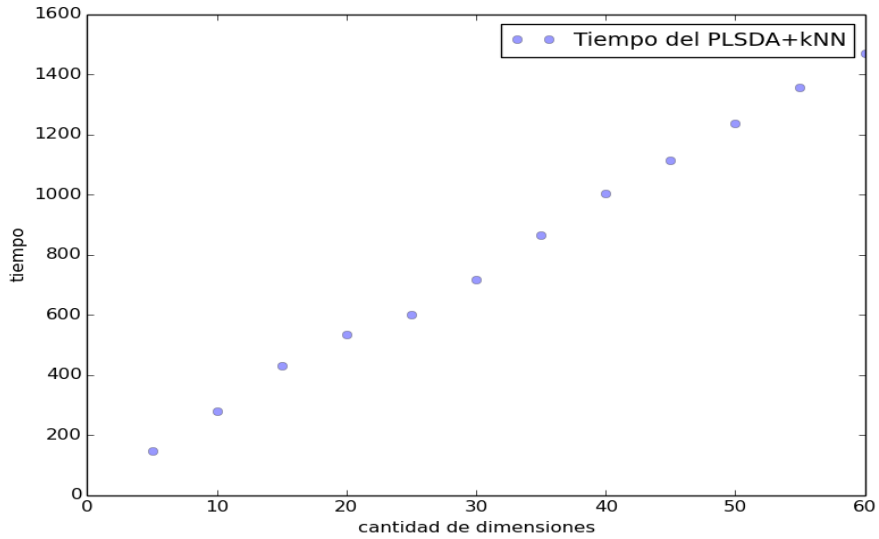
Con este experimento determinamos que la combinación que mejor nos resulta para utilizar el método PLSDA es con 55 dimensiones y 7 vecinos a contemplar por el método kNN.

Para observar un análisis más completo del experimento se realizó uno extra variando dimensiones entre 1 y 50 aumentando de a 1 en cada iteración, con los siguientes resultados.



En el gráfico se puede observar el crecimiento del hitrate a medida que aumenta la cantidad de dimensiones y cómo la pendiente cae a medida que se acerca a la dimensión 10, viendo así que en las primeras dimensiones se encuentra la mayor cantidad de información. Es por esto que podemos considerar que el rango de valores utilizados en el experimento previo (5, 60) fue representativo para poder encontrar un valor que nos funcione bien con el método (que nos de buenos resultados) dado que la curva se mantiene a partir del valor 10.

- Las mediciones respecto del *tiempo* (en segundos) fueron las siguientes:



Podemos observar como, aunque el método funcione mejor con más dimensiones, se paga un costo extra en tiempo de cómputo. Podemos observar a partir del gráfico la relación lineal que ejerce la variable *dimensiones* sobre la complejidad temporal del método PLSDA.

4.3. Tercer Experimento

Ya fijadas la cantidad de vecinos en 7 y cantidad de dimensiones en 55, valores que nos sirven para categorizar mejor las imágenes, en el siguiente experimento nos proponemos variar la cantidad de *folds* del método *k-fold cross validation*. En los anteriores experimentos usamos K igual a 10. Como mencionamos en la sección de desarrollo, veremos en esta oportunidad K igual a 5 y 15 para realizar las mediciones estadísticas. Los resultados fueron los siguientes:

- Las mediciones respecto del *hit rate* fueron las siguientes:

hit rate	K
97,221 %	5
97,329 %	10
97,321 %	15

- Las mediciones respecto del *tiempo* fueron las siguientes:

tiempo	K
1380.03	5
1356.84	10
1349.84	15

Estas mediciones de tiempo están hechas en segundos.

Respecto de estos resultados concluimos a partir de este experimento que el K que mejor funciona para nuestros métodos es 10. Fue el valor que nos proveyó el mejor *hit rate*, a su vez que en términos de tiempo de cómputo no hubo una diferencia sustancial entre los 3.

Por último, con este experimento determinamos que la combinación que mejor funcionó para el método PLSDA fue:

- Cantidad de vecinos: 7.
- Cantidad de dimensiones: 55.
- K: 10.

4.4. Cuarto Experimento

En este experimento nos propusimos averiguar cómo influye en el categorizador variar la cantidad de componentes α en el método PCA. Para esto, fijamos la cantidad de vecinos utilizados por kNN en 5 y variamos las componentes de 5 a 50 incrementando de a 5. Nuestra suposición es que a menor cantidad de componentes menor será la calidad del categorizador, ya que más información relevante será descartada. De todas maneras, esto será hasta un cierto límite, ya que al agregar demasiada información puede ser que se agregue más ruido.

Con este experimento podemos decidir cuántas componentes como mínimo deben ser utilizadas para que los resultados sean deseables. También se puede analizar la relación calidad-complejidad temporal.

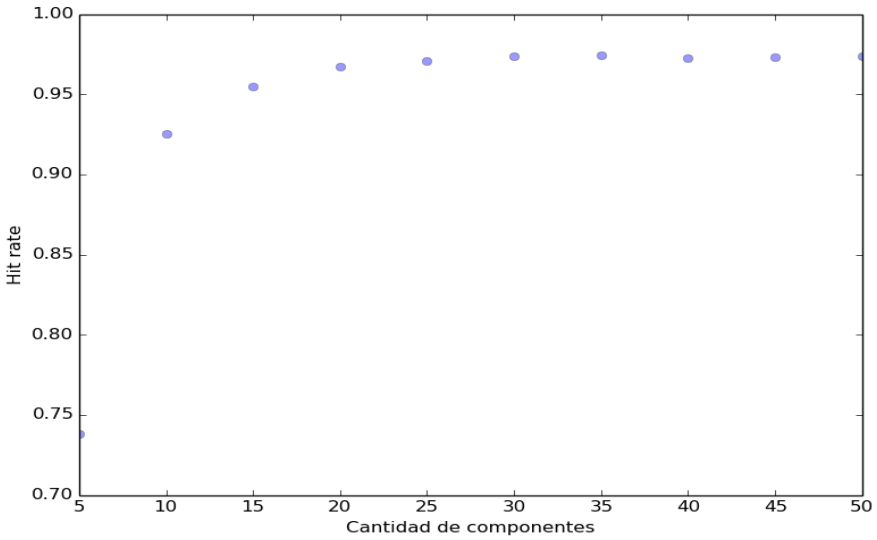


Figura 1: Hitrate de PCA variando α

Como se puede observar en el gráfico 1 para esta prueba, a partir de las 35 componentes no se nota una diferencia significativa en el *hitrate*. De 5 a 20 componentes se ve un aumento pronunciado (creemos que esto se debe a que con tan pocas componentes se pierde información sustancial para catalogar), y a partir de ahí oscila entre 96,4 % y 97,3 % porcentaje de aciertos pero no se ve un aumento creciente.

El incremento de tiempo del mismo método pero con 40, 45 y 50 componentes es:

componentes	aumento de tiempo*
40	5,92 %
45	10,6 %
50	15,3 %

*El porcentaje de incremento es con respecto al tiempo de corrida de 35 componentes, que tardó 2205,9 segundos.

Debido a la relación tiempo-calidad de la solución, y notando que a medida que crecen las componentes crece el tiempo de ejecución, decidimos que 35 componentes es un buen valor para el categorizador bajo esta condiciones.

En el siguiente gráfico (ver 2) , podemos ver cómo la precisión promedio está por encima de 95 % a partir de las 15 componentes, llegando a la máxima en 35 y manteniéndose a partir de ahí. Este es otro motivo por el cual nos parece que 35 componentes es un buen valor para fijar la variable cantidad de componentes (α).

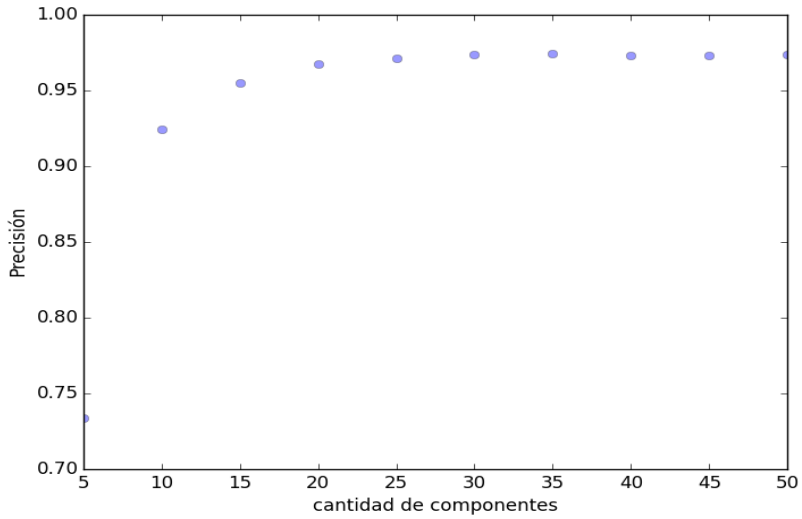


Figura 2: Precisión de PCA variando α

No tuvimos ningún motivo en particular para creer que el categorizador predeciría mejor un número determinado por sobre otro, y como se puede observar en 3 , donde cada punto indica la precisión para cada dígito, es que dicha precisión está en un rango similar para todos los valores para cada α.

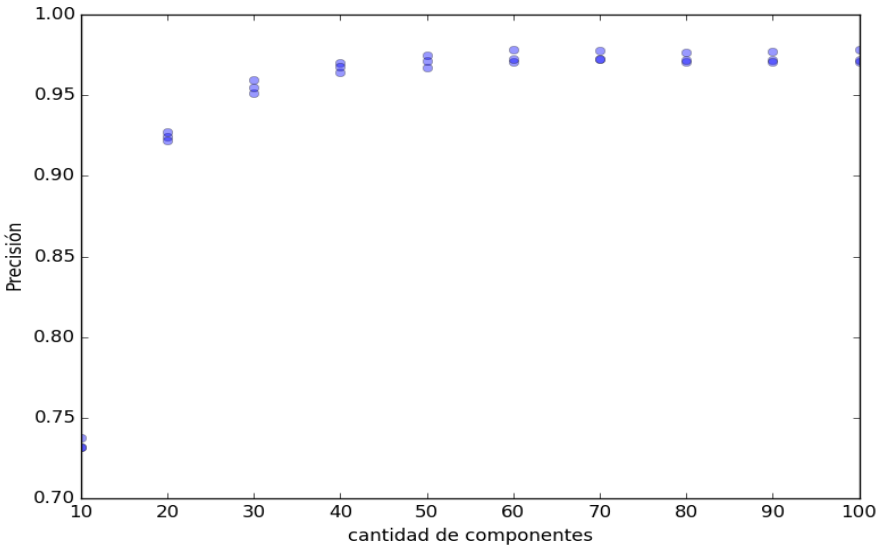


Figura 3: Precisión de cada dígito variando α

4.5. Quinto Experimento

Luego de haber encontrado la cantidad de componentes que devuelve resultados deseables, decidimos experimentar sobre la cantidad de vecinos, para ver si chequear contra más vecinos mejora la calidad del categorizador. Creemos que al aumentar mucho la cantidad de vecinos, va a tener en cuenta para su categorización una porción notable de la muestra entera que puede hacer que se catalogue con un valor erróneo. Por esta razón, fijamos las componentes en 35 y variamos la cantidad de vecinos de 5 a 25, incrementando de a 5.

Como se puede ver en 4 , efectivamente el mejor hitrate alcanzado es con 5 vecinos:

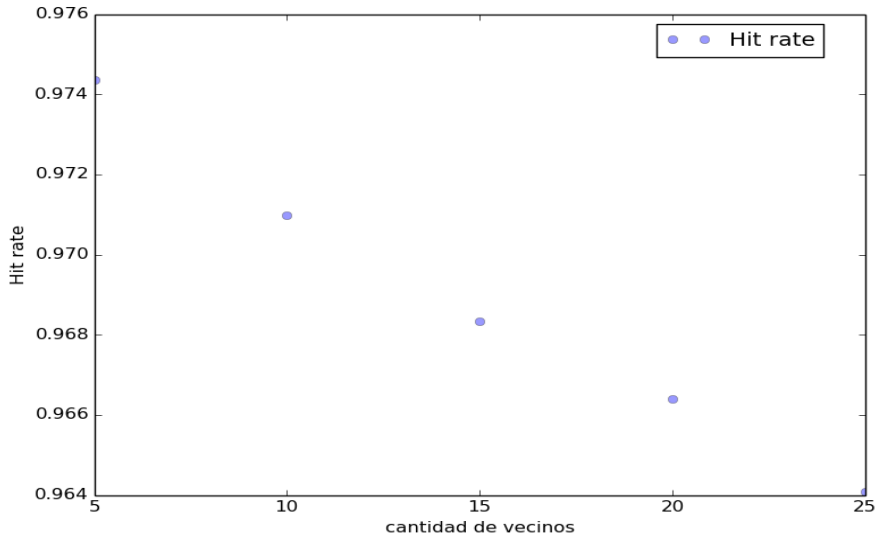


Figura 4: Hit rate de PCA variando k con $\alpha = 35$

Y, además, a medida que crece la cantidad de vecinos, la cantidad de imágenes bien catalogadas decrece.

Con la precisión ocurrió lo mismo, a medida que aumentan los vecinos, ésta decrece. Creemos que el motivo es el mismo, si se compara contra más vecinos cercanos, pueden entrar en consideración otros con

otra etiqueta que termine haciendo que el valor se catalogue como ellos, en lugar de catalogarse como los más cercanos. Esto se puede ver en la siguiente imagen.

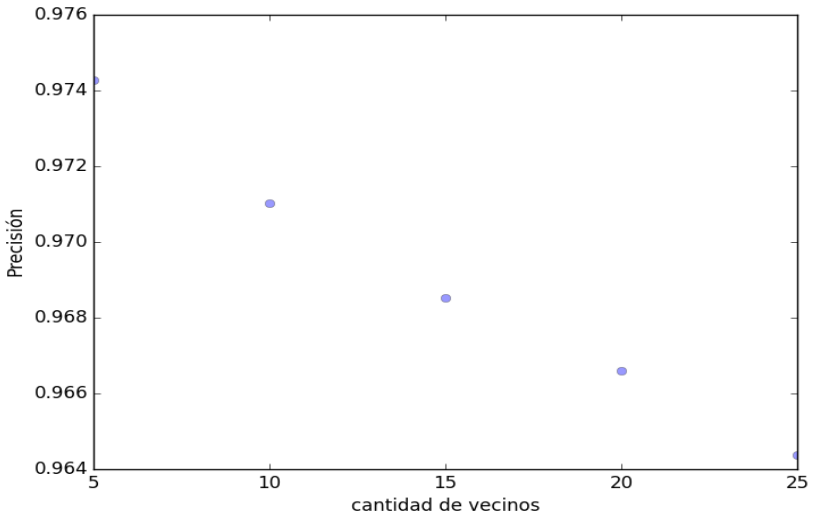


Figura 5: Precisión de PCA variando k con $\alpha = 35$

Contrario a lo que ocurrió en el experimento anterior, puede verse una variación notable en la precisión de cada dígito para cada cantidad de vecinos k analizada (ver 6). Creemos que esto de debe a que a mayor cantidad de vecinos, como vimos en los gráficos anteriores, peor es la clasificación global, y puede ocurrir que determinados dígitos sean peor catalogados que otros por la forma en la cual están distribuidos.

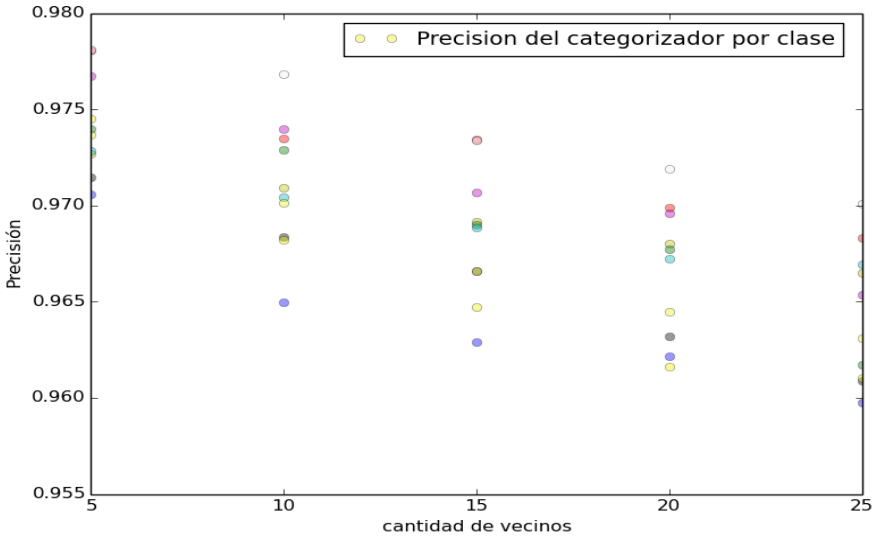


Figura 6: Precisión de cada dígito variando k

Como la cantidad de vecinos obtenida fue en uno de los extremos del intervalo, decidimos aumentar la granularidad y experimentar en un rango más cercano al 5. Nos propusimos examinar el rango [1..7] aumentando la cantidad de vecinos de a 1.

Los resultados fueron los siguientes:

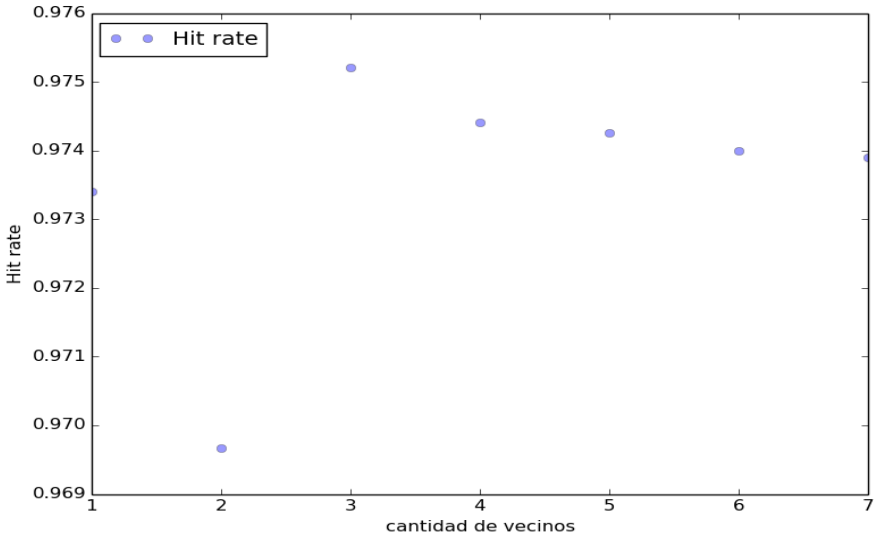


Figura 7: Hit rate de PCA variando k con $\alpha = 35$

Efectivamente, la cantidad de vecinos que nos maximiza el hitrate es 3, dando un 97,52 % de hit rate.

En este experimento creemos que ocurre algo similar al primer experimento cuando se analizan dos vecinos.

4.6. Sexto Experimento

Ya fijadas la cantidad de vecinos en 3 y cantidad de dimensiones en 35, valores que nos sirven para categorizar mejor las imágenes, en el siguiente experimento nos proponemos variar la cantidad de *folds* del método *k-fold cross validation*. En los anteriores experimentos usamos K igual a 10. Como mencionamos en la sección de desarrollo, veremos en esta oportunidad K igual a 5 y 15 para realizar las mediciones estadísticas. Los resultados fueron los siguientes:

- Las mediciones respecto del *hit rate* fueron las siguientes:

hit rate	K
97,39 %	5
97,52 %	10
97,5 %	15

- Las mediciones respecto del *tiempo* fueron las siguientes:

tiempo	K
1031.92	5
1106.94	10
1226.56	15

Estas mediciones de tiempo están hechas en segundos.

Respecto de estos resultados concluimos a partir de este experimento que el mejor K para usar con este métodos es 10. Fue el valor que nos proveyó el mejor *hit rate* y además, el tiempo de cómputo requerido por los 3 no fue sustancialmente diferente.

Por último, con este experimento determinamos que la combinación que mejor funcionó para el método PCA fue:

- Cantidad de vecinos: 3.
- Cantidad de dimensiones: 35.
- K: 10.

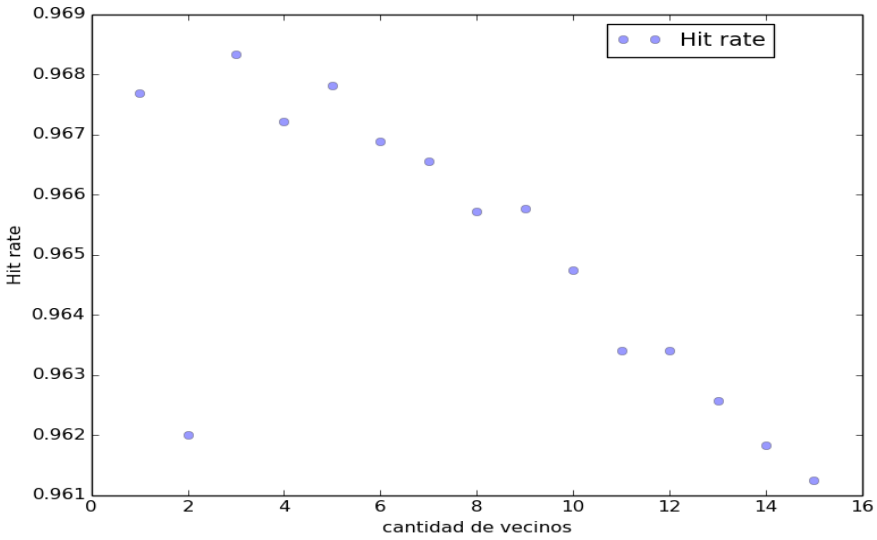
4.7. Séptimo Experimento

Finalmente nos propusimos ver qué resultados nos ofrece el método kNN sin reducción de dimensión.

Para esto decidimos hacer un experimento variando la cantidad de vecinos de 1 a 15 aumentando en cada iteración de a 1. Si nuestras hipótesis son correctas el algoritmo debería mostrar su eficiencia/complejidad en este rango de forma significativa.

Los resultados del experimento fueron los siguientes:

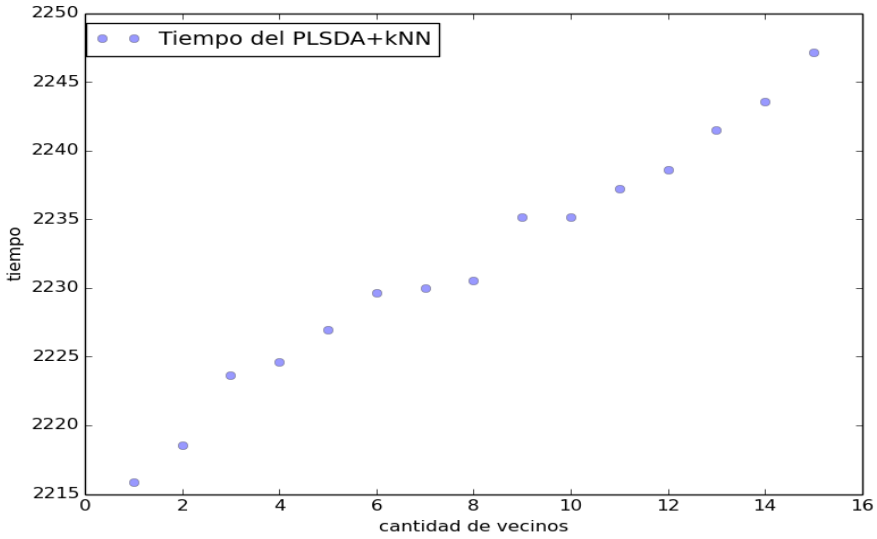
- Las mediciones respecto del *hit rate* fueron las siguientes:



  Podemos observar un claro pico al considerar sólo 3 vecinos y cómo al aumentar la cantidad de vecinos decrece la cantidad de los que fueron clasificados correctamente. Como mencionamos previamente en otros experimentos, suponemos que la razón de esto es que al considerar un vecindario más grande se puede agregar más *ruido* a la estimación.

  Otra observación que se puede realizar con respecto a este gráfico, es que al tomar cantidad de vecinos impar, el hitrate es mayor (e.g. el hitrate con un vecino es mayor que con dos, con tres es mayor que con cuatro, con cinco es mayor que con seis). Creemos que esto se debe a que si la cantidad de vecinos analizados es par, el desempate (que se realiza por mayor valor númerico de la clase) hace que se pierda fidelidad en la clasificación, en contraposición a cuando la cantidad de elementos es impar, donde la clasificación del valor estará dada por la clasificación de los vecinos cercanos que más veces aparezcan.

- Las mediciones respecto del *tiempo* fueron las siguientes:



Podemos observar la relación lineal que ocurre entre cada iteración al aumentar la cantidad de vecinos.

En base a este experimento deducimos que la cantidad de vecinos que mejor funcionó con nuestro método es 3. No sólo porque a medida que aumentamos la cantidad de vecinos disminuía el *hit rate* sino que además el tiempo de cómputo requerido era mayor.

4.8. Octavo Experimento

En este experimento nos proponemos, una vez encontrada la cantidad de vecinos que mejor funcionaba con nuestra implementación de kNN, variar la cantidad de *folds* utilizados para entrenamiento y prueba, y así ver si varía la eficiencia del método. Dicha cantidad de vecinos había sido 3. Los resultados fueron los siguientes:

- Las mediciones respecto del *hit rate* fueron las siguientes:

hit rate	K
96,66 %	5
96,83 %	10
96,82 %	15

- Las mediciones respecto del *tiempo* fueron las siguientes:

tiempo	K
1031.92	5
1539.56	10
2223.94	15

Como podemos observar, el valor que nos proveyó un mejor *hit rate* es K igual a 10. De todas maneras, dependerá del usuario de este método ponderar entre el tiempo que le requiera utilizarlo, respecto del *hit rate* esperado. Por ende, quizás se podría preferir utilizar K igual a 5, dado que la diferencia de *hit rate* no es mucho menor, pero el tiempo de cómputo requerido sí.

De todas maneras, como nuestro interés es mejorar la cantidad de imágenes bien estimadas, los valores que mejor nos resultan para el método de kNN sin preprocesamiento son:

- Cantidad de vecinos: 3.
- K: 10.

5. Discusión

En esta sección nos proponemos comparar y discutir respecto de la mejor configuración de valores hallados para los tres métodos implementados:

- kNN
- PLSDA + kNN
- PCA + kNN

Dichos valores fueron:

- kNN:
  - Cantidad de vecinos: 3
  - K = 10
- PLSDA + kNN:
  - Cantidad de vecinos: 7
  - Cantidad de dimensiones: 55
  - K = 10
- PCA + kNN:
  - Cantidad de vecinos: 3
  - Cantidad de dimensiones: 35
  - K = 10

Como lo que nos propusimos mejorar a través de estos métodos es el porcentaje de imágenes bien estimadas por el clasificador (*hit rate*) veremos cual fue el que mejor nos resultó:

Método	Hit rate
kNN	96,83 %
PCA + kNN	97,52 %
PLSDA + kNN	97,329 %

Concluimos que el método que, en base a nuestros experimentos, nos proveyó los mejores resultados fue PCA.

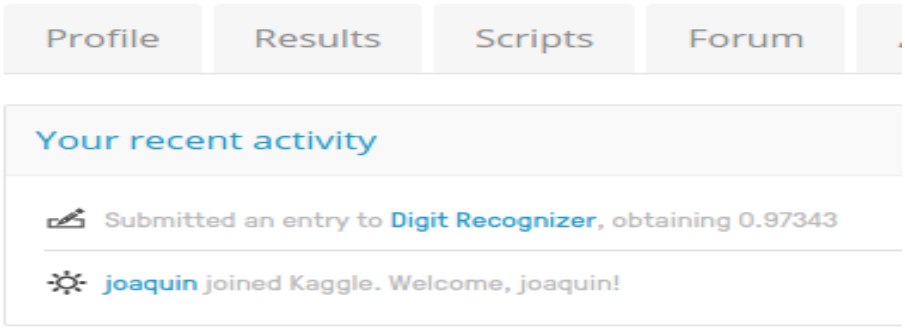
Si bien PLSDA provee más información a la hora de realizar las predicciones por incluir la clase a la que pertenece cada uno, no logró superar la performance de PCA. Por otro lado la diferencia entre ellos es muy poca, nuestra hipótesis es que puede deberse al set de entrenamiento usado, como el método depende fuertemente del mismo para encontrar sus valores óptimos, también debe variar los resultados que pueden ser obtenidos con ambos métodos. Con este análisis ninguno es necesariamente mejor que el otro, simplemente dependerá del caso en el que se use.

Creemos que la diferencia significativa de PCA contra kNN está dada por las trasformaciones que realiza el primer método, que eliminan los datos innecesarios o que generan ruido, y que ahí radica el por qué de la diferencia entre ambos.

6. Kaggle

El desarrollo de los experimentos explicados previamente son utilizando como base de datos un conjunto de imágenes de las cuales se sabe, para cada una, a qué clase pertenece (qué dígito es). Como mencionamos previamente, la cantidad de imágenes provistas para estos experimentos fueron 42000, de las cuales, para cada una, se sabe qué dígito le corresponde. Además contamos con un conjunto de 28000 imágenes de las cuales no sabemos a qué dígito corresponden. Una de las tareas a realizar en este trabajo fue, una vez encontrados los valores que mejor funcionen para nuestra implementación de los métodos, elegir el mejor de ellos (el qué mejor resultados nos proveyó) y aplicar el método entrenando con todo el conjunto de imágenes inicial, y estimar sobre el conjunto del cual no sabemos, para ninguna imagen, su etiqueta.

Entrenamos al clasificador con las 42000 imágenes utilizando el método PCA con 35 componentes y 3 vecinos, y estimamos para cada una de las 28000 imágenes, qué dígito le corresponde. Para saber cuan bien funcionó este método con estos datos aplicamos a la competencia de la cual se desprende este trabajo en la pagina www.kaggle.com, de donde obtuvimos los datos en un principio. Los resultados fueron los siguientes:



Tiempo de cómputo requerido: 1983.474767

7. Conclusiones

En este trabajo analizamos e implementamos un método simple aplicado a determinar a qué valor pertenece un dígito de una imagen dada, así como dos métodos para reducir la llamada "maldición de la dimensión"que este sufre, haciendo uso de algunas hipótesis que el problema conllevaba. El mismo probó ser bastante poderoso para resolver problemas de esta índole con la contra de necesitar bastante tiempo de procesamiento.

Dado todo el análisis del método, ¿Hasta qué punto tiene sentido usarlo con o sin las reducciones de dimensión? ¿Qué tan efectivo es aplicado a otro problema en la práctica?. Si bien las respuestas dependen del contexto con el que vaya a ser usado, podemos decir que en general para datos no demasiado grandes[1] y, haciendo un análisis previo sobre los parámetros cercanos a los óptimos de la base de entrenamiento, el método presenta una herramienta sencilla y poderosa para clasificar datos.

Tras finalizar el trabajo surge la pregunta ¿A qué otros problemas se puede aplicar? Si bien se aplicó a reconocimiento de dígitos, siempre que el problema que se quiera clasificar pueda ser representado con el modelo propuesto, el método podrá clasificar (en mejor o peor forma) el problema dado. Hay que resaltar esta propiedad del método ya que se abstrae de la semántica del problema a resolver y solo se concentra en las relaciones que contienen los datos ingresados.

[1]Para los experimentos se usaron imágenes con 768 píxeles en escalas de grises, tener en cuenta que una de 100x100 píxeles tiene 10^4 puntos.