



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico II

CSI: DC

Métodos Numéricos  
Primer Cuatrimestre de 2016

Integrante	LU	Correo electrónico
Cortés, Lucas	302/13	lucascortes@me.com
Lamela, Emanuel	021/13	emanuel93_13@hotmail.com
Zimenspitz, Ezequiel	155/13	ezeqzim@gmail.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

Este trabajo ahonda en técnicas de clasificación de dígitos manuscritos digitalizados. Fundamentado con postulados y métodos de las áreas de análisis numérico y *machine learning*, nos embarcamos en querer ponderar y extraer la información más relevante de las imágenes para su reconocimiento y categorización.

## Palabras claves

- *Machine Learning*
- Reconocimiento de dígitos manuscritos
- Análisis de componentes principales
- Análisis discriminante con cuadrados mínimos

## Índice

<b>1. Introducción Teórica</b>	<b>3</b>
1.1. k Vecinos Más Cercanos ( <i>kNN</i> )	3
1.2. Consideraciones previas a <i>PCA</i> y <i>PLS-DA</i>	4
1.3. Análisis por Componentes Principales ( <i>PCA</i> )	4
1.4. Análisis Discriminante por Cuadrados Mínimos Parciales ( <i>PLS-DA</i> )	5
<b>2. Desarrollo</b>	<b>6</b>
2.1. Metodologías de Clasificación	6
2.1.1. Clasificación <i>naive</i> con <i>kNN</i>	6
2.1.2. Reducción de dimensión <i>no</i> supervisada con <i>PCA</i>	6
2.1.3. Reducción de dimensión supervisada con <i>PLS-DA</i>	7
2.1.4. Clasificación inteligente: <i>Transformación</i> + <i>kNN</i>	7
2.2. Estrategias de medición	8
2.2.1. Evaluación robusta con <i>K-fold cross validation</i>	8
2.2.2. Métricas de calidad	8
2.3. Algoritmos de Utilidad	9
2.3.1. Método de la Potencia	9
2.3.2. Deflación	9
<b>3. Resultados y discusión</b>	<b>10</b>
3.1. Análisis Previo: Alfa y Gama	10
3.2. El nuevo espacio	12
3.3. Análisis de <i>KNN</i>	20
3.3.1. Costo temporal	20
3.3.2. Calidad del algoritmo	20
3.4. Análisis de <i>PCA</i> y <i>PLS-DA</i> con <i>kNN</i>	26
3.4.1. Descripción	26
3.4.2. <i>PCA</i>	26
3.4.3. <i>PLS-DA</i>	28
3.4.4. Cross Validation	31
<b>4. Conclusiones</b>	<b>32</b>
<b>5. Extras</b>	<b>33</b>
5.1. Veredicto de <i>Kaggle</i>	33
5.2. Alternativa de clasificación: <i>Promedio</i> de imágenes	33
<b>6. Referencias</b>	<b>34</b>

## 1. Introducción Teórica

El problema de reconocer e identificar imágenes es uno que, a simple vista, uno creería que depende necesariamente del ojo humano. El simple hecho de plantearse cómo una computadora pudiese decidir algo del estilo “esta imagen es de un perro” parece de por sí increíble e, inclusive, extremadamente *difícil*.

Lo interesante de las problemáticas es que uno puede *encararlas* de una u otra manera que puede abrir el camino para resolverlo. En el caso de asociación de imágenes, uno puede pensarlo como *reconocer* o, en contraposición, *diferenciar* una ilustración. En este trabajo vamos a atacar el problema (no este sino uno más *acotado*), por ambos caminos para poder arribar a una conclusión.

Dicho esto, poder decidir para *cualquier* tipo de imagen, sin ningún contexto, a qué objeto o concepto está asociado sigue siendo de dificultad elevada. El análisis del contexto es esencial para cualquier tipo de problema y este no es la excepción.

La situación concreta a ser abarcada en este trabajo es la de reconocer dígitos en imágenes con números manuscritos con un formato particular:

- El número escrito debe ser del 0 al 9
- El tamaño es de  $28 \times 28$  píxeles
- Está en escala de grises de valores entre 0 (*negro*) y 255 (*blanco*)

Uno podría argumentar que es un formato acotado, pero si uno digitaliza un texto con números manuscritos, en general puede partirlo en imágenes con las condiciones pedidas. Hay escenarios en los cuáles este problema tiene relevancia, por nombrar algunos:

- Reconocimiento de textos antiguos
- Validación digital de textos
- Reconocimiento de caligrafía

Ahora ahondaremos en los fundamentos teóricos de los *métodos numéricos* que servirán de base para el desarrollo de este trabajo.

### 1.1. k Vecinos Más Cercanos (*kNN*)

Un primer *approach* planteado es el de ***k-Nearest-Neighbours***. Como modelos tenemos:  $C$  un conjunto de clases,  $A = \{a_1, \dots, a_n\}$  y  $B = \{b_1, \dots, b_n\}$ ,  $a_i, b_i \in \mathbb{R}^m$ , dos conjuntos de elementos que pueden ser clasificados o *taggeados* como elementos de  $C$  con la particularidad de que conocemos la clasificación de elementos del conjunto  $B$ . Por lo tanto, el problema va a ser *taggear* el conjunto  $A$  en base a los de  $B$ . El algoritmo es simple:

Consideramos  $a \in A$ , y una función  $d : A \times B \mapsto \mathbb{R}_{\geq 0}$  que denominaremos **función de distancia**. Esta función toma un elemento de  $A$  y otro de  $B$ , y devuelve un valor que *resume* las diferencias entre un parámetro y el restante a un valor. Utilizaremos la misma para calcular las “distancias” de  $a$  contra todos los elementos  $b \in B$ , quedandonos sólo con los  $k$  de menor valor. Estos serán los que más se *parezcan* a  $a$ .

Una vez realizado esto, sea  $f : C \times B \subset B \mapsto \mathbb{N}$  una función que nos devuelve la cantidad de veces que se encuentra un *tag* dentro de un sub-conjunto de  $B$ , tomamos  $c = \operatorname{argmax}_{c \in C} f(c, K)$ . De esta forma, “asignamos” la clase  $c$  a  $a$ .

Este algoritmo efectivamente realiza una **clasificación supervisada**, aprendiendo de una base de “entrenamiento” para categorizar los elementos restantes.

De todas formas, lo simple del algoritmo tiene su contraparte en la variable tiempo de ejecución. Uno debe aplicar  $d$  sobre todos los elementos de  $B$ , con lo cuál si el conjunto es grande, dependemos de la complejidad temporal la función de distancia. La clave de este procedimiento es elegir una función que resume las diferencias entre elementos más precisamente, minimizando el computo, lo cuál no es una tarea fácil ya que, en general, funciones de distancia livianas representan menor calidad de resultados.

En respuesta a esa problemática, planteamos 2 métodos denominados **Análisis por Componentes Principales (PCA**, por sus siglas en inglés) y **Análisis Discriminante por Cuadrados Mínimos Parciales**

(PLS-DA). Conceptualmente, consisten en ponderar la información para decidir qué *variables* almacenan más información, y usarlas para reducir el análisis a ese conjunto de variables de manera tal que se mejore dramáticamente el tiempo de cómputo.

## 1.2. Consideraciones previas a PCA y PLS-DA

Tanto PCA y PLS-DA tienen como pilar realizar un *cambio de base* conveniente. En cada método varía *cuál se busca*.

Nos abstraemos del modelo y consideramos  $x_i$ ,  $i = 1, \dots, m$ , como  $m$  variables aleatorias, y  $x^j \in \mathbb{R}^m$ ,  $j = 1, \dots, n$ , como  $n$  muestras de las mismas. Además,  $\mu = (x^{(1)} + \dots + x^{(n)})/n \in \mathbb{R}^m$  la media entre las mismas. Notar que  $\mu_i = (x_i^{(1)} + \dots + x_i^{(n)})/n$  es el promedio de la variable aleatoria  $i$  y con la cuál podemos reescribir  $\mu$  como  $\mu = (\mu_1, \dots, \mu_n)$ .

Planteamos la transformación sobre la matriz  $X$ , la cuál cuenta con  $n$  filas de la forma  $(x^{(i)} - \mu)^t$ ,  $i = 1, \dots, n$ . Esta es la matriz de datos que efectivamente centra la media de las variables en 0, lo cuál es simple (y se lo dejamos al lector) de verificar realizando el cálculo de la media sobre cada columna de  $X$ . Además, facilita los pasos aritméticos y algebraicos que se deben llevar a cabo en los métodos.

La transformación la caracterizaremos como  $P$  y la buscamos tal que  $\hat{X}^t = PX^t$ , donde  $\hat{X}^t$  vive en un **universo de mayor conveniencia para el análisis**. Adicionalmente, queremos que  $P$  sea ortogonal por el comportamiento que posee de no alterar el largo de los vectores a los que multiplica, para que no modifique el comportamiento de los datos.

## 1.3. Análisis por Componentes Principales (PCA)

En este primer caso, **planteamos una transformación que simultáneamente maximice las varianzas y minimice las covarianzas** entre las variables aleatorias. Lo que logramos con esto es reducir las dependencias, para poder analizar una variable sin considerar qué ocurre con las demás, y extraer la mayor cantidad de información de cada una.

Una forma es pensarlo matricialmente. Llamamos la matriz  $M_X$  como **la matriz de covarianzas**, que posee en la posición  $(j, k)$  la covarianza entre las VA  $x_j$  y  $x_k$ , que denominamos  $\sigma_{x_j x_k}$ . Dicha matriz se puede obtener haciendo  $M_X = \frac{1}{n-1} X^t X$ . Para verificarlo desarrollamos  $M_X^{(j,k)}$ , con  $j, k \in 1, \dots, n$ :

$$M_X^{(j,k)} = \frac{1}{n-1} \text{fila}_j(X^t) \text{col}_k(X)$$

$$\begin{cases} \text{fila}_j(X^t) = (x_j^{(1)} - \mu_j, \dots, x_j^{(n)} - \mu_j)^t = ((x_j^{(1)}, \dots, x_j^{(n)}) - \mu_j v)^t = (x_j - \mu_j v)^t \\ \text{col}_k(X) = (x_k^{(1)} - \mu_k, \dots, x_k^{(n)} - \mu_k) = ((x_k^{(1)}, \dots, x_k^{(n)}) - \mu_k v) = (x_k - \mu_k v) \end{cases}$$

Dónde  $v = (1, \dots, 1) \in \mathbb{R}^n$ .

Por lo tanto,

$$M_X^{(j,k)} = \frac{1}{n-1} \text{fila}_j(X^t) \text{col}_k(X) = (x_j - \mu_j v)^t (x_k - \mu_k v) / (n-1) = \sum_{i=1}^n (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k) / (n-1) = \sigma_{x_j x_k}^1.$$

Observamos además que  $\sigma_{jj} = \sum_{i=1}^n (x_j^{(i)} - \mu_j)^2 / (n-1) = \sigma_j^2$ . Dejamos en manos del lector verificar que  $M_X^{(j,k)} = M_X^{(k,j)}$ . Con lo cuál obtuvimos una matriz que contiene a las varianzas de las variables en la diagonal y las covarianzas en el resto de las posiciones, y que es simétrica. Esta última observación va a ser fundamental para los próximos pasos del procedimiento.

Retomemos el cambio de base y la búsqueda de la transformación  $P$ . Veamos como podemos obtenerla a partir de  $M_{\hat{X}}$ , la matriz de covarianzas de  $\hat{X}$ .

$$M_{\hat{X}} = \frac{1}{n-1} \hat{X}^t \hat{X} = \frac{1}{n-1} (PX^t)(XP^t) = P \frac{X^t X}{n-1} P^t = P M_X P^t.$$

<sup>1</sup>No es la varianza *teórica*, sino un *estimador insesgado* de la misma. Dado que no analizamos las variables aleatorias en términos teóricos sino más bien en muestras, consideramos que es lo suficientemente preciso

Sabiendo que  $M_X$  es simétrica sabemos que **existen  $V$  ortogonal, cuyas columnas son sus autovectores, y  $D$  diagonal tal que  $M_X = VDV^t$**  que sabemos que existen [1]. Proponemos  $P = V^t$ , y arribamos a que  $M_{\hat{X}} = V^t M_X V = (V^t V) D (V V^t) = D$ , ya que  $V$  es ortogonal. El conjunto de autovectores que forman  $V$  son denominados **Componentes Principales** de los datos.

Por lo tanto, la transformación propuesta es  $P = V^t$ . Como comentario final adicional, agregamos que este resultado se distancia completamente del concepto de clases o categorías mencionado en la sección anterior, por ende por sí mismo es un método **no supervisado**.

#### 1.4. Análisis Discriminante por Cuadrados Mínimos Parciales (PLS-DA)

En contraposición a *PCA*, el siguiente método es **supervisado**. Lo que le brinda esa distinción es que **utiliza las clases (o etiquetas)**.

Si bien el método se denomina *PLS-DA*, el fundamento se encuentra en **PLS regression** (Regresión por Cuadrados Mínimos) al cuál luego se le aplica el criterio que forma el método final. El *PLS* tradicional corresponde a un modelo *binario* de clasificación, cada muestra se encuentra o no en una clase determinada. **La extensión discriminante permite un análisis multi-categorico como el planteado en este trabajo**. Más allá de esto, primero desarrollaremos el método tradicional.

Siendo  $X$  tal como fue definida en *PCA*, definimos  $Y$  como un vector que posee para cada posición  $i$  la información sobre a *que* clase o *tag* pertenece la imagen  $i$ , dónde el índice se corresponde con la imagen  $i$  en  $X$ . **La transformación busca maximizar las covarianzas entre las variables y las etiquetas en el nuevo espacio**. Este criterio va a permitir **extraer la información relevante que relacione muestra con clase**. La obtendremos a partir del siguiente planteo:

$$\begin{cases} X = TR + E \\ Y = UQ + F \end{cases}$$

Este desglose se puede pensar como:  $T$  nos *transporta* al nuevo universo de los datos,  $R$  lo revierte, y finalmente, como se puede sufrir pérdida de información, sumamos  $E$  de tal forma de mantener la consistencia. Una construcción análoga se puede hacer para  $Y$ . Sólo nos incumbe poder hacer *el viaje de ida* para luego extraer información en ese *espacio alterno*, por lo tanto nos concentraremos exclusivamente en  $T$  y  $U$ .

Si  $t$  y  $u$  son vectores de las *matrices de transformación*  $T$  y  $U$ , respectivamente, quiero maximizar su covarianza, para lo que utilizaremos el siguiente resultado:

$$Cov(t, u)^2 = Cov(Xw, Yc)^2 = \max_{||r||=||s||=1} Cov(Xr, Ys)$$

Como  $t$  y  $u$  *viven* en el nuevo espacio de datos, los podemos pensar como  $Xw$  y  $Yc$ . Utilizando esta cadena de igualdades, se puede ver que el  $w$  **que cumple esto es el autovector dominante de  $X^t Y Y^t X$** . Con este razonamiento, se puede plantear un esquema iterativo (que está explícito en 2.1.3) de manera tal de extraer tantos vectores  $w$  como se pida para obtener  $P$  (la matriz de cambio de base de  $X$ ) formada por los susodichos.

Finalmente, como queremos poder trabajar con variables *multi-categoricas* como son las de este trabajo (en otras palabras, pasar de *PLS* a *PLS-DA*), **redefinimos  $Y$  de tal manera que cada columna posea información sobre cada clase**.

## 2. Desarrollo

El suelo está labrado en cuestiones teóricas. En esta sección nos avocaremos en explicar el papel que juega cada método y, el proceso completo partiendo de las imágenes y concluyendo en su clasificación.

Vamos a considerar las imágenes, siendo  $n \in \mathbb{N}$  la cantidad, como  $x^{(i)} \in \mathbb{R}^m$  con  $m = 28 \times 28 = 784$  y  $i \in \{1, \dots, n\}$ . Al conjunto de imágenes lo denominamos  $I$ . Como están en escala de grises, además se cumple que  $x_j^{(i)} \in \{0, \dots, 255\}$ , donde  $x_j^{(i)}$  es el  $j$ -ésimo elemento de  $x_i$ ,  $\forall j \in \{1, \dots, 784\}$ . En términos coloquiales, una *tira* de 784 valores que están en el rango de 0 a 255. Adicionalmente, el conjunto  $C = \{0, \dots, 9\}$  es el conjunto de clases/labels/tags/dígitos, según como los denominemos en cada ocasión particular. Finalmente, por ahora, vamos a considerar una partición de  $I = A \cup B$ , donde  $A$  e  $B$  son disjuntos y los mismos tienen las particularidades destacadas en 1.2.

### 2.1. Metodologías de Clasificación

#### 2.1.1. Clasificación *naïve* con kNN

En una primera aproximación, **utilizamos el kNN como método de clasificación a secas**, lo que es decidir a qué dígito pertenece cada imagen del conjunto  $A$ . Tomamos  $a \in A$  una tira que buscamos *taggear*.

Recordar que, por cómo lo definimos en la *sección 1.1*, el algoritmo requería **una función de distancia**  $d$ . Como modelamos con vectores, proponemos la **norma 2**. En otras palabras, siendo  $z$  y  $x$  dos imágenes  $d(z, x) = \|z - x\|_2^2 = (z - x)^t(z - x)$ . Notar que en su forma de *producto interno*, el cómputo no pierde precisión por ser suma y multiplicación de números entre 0 y 255. Adicionalmente, fue elegida por el nivel de precisión que posee (en definitiva, compara una a una cada componente).

En consecuencia de haber definido una  $d$ , **obtenemos el conjunto de  $k$  vecinos más cercanos**. Clasificar es fácil: se cuentan las etiquetas de cada uno de los  $k$  vecinos y la que más se repita, se le asigna a  $a$ <sup>2</sup>.

Desafortunadamente, **la simplicidad tiene su costo temporal en este caso**. Las imágenes tienen 784 pixels, es decir, cada punto a considerar tiene 784 componentes. Calcular la distancia de un punto de esta dimensión contra  $|B|$  (el tamaño de  $B$ ) de la misma dimensión suena a mucho trabajo y *lo es*. Aquí es donde cobran valor **PCA** y **PLS-DA**.

Ambos tienen la misma idea, obtener una matriz que realice un cambio de base tal que permita quedarnos con sólo una *porción*, la de mayor contenido de información, de la misma. Aunque por sí solos, no son métodos de categorización. Por ende, **estarán involucrados como preprocesadores de la información a ser servida al kNN** (reducir las dimensiones a considerar previo a aplicar kNN).

#### 2.1.2. Reducción de dimensión no supervisada con PCA

Siguiendo la línea de **PCA** (1.3), buscamos  $P$  conformado por las **Componentes Principales** que son los autovectores de  $M_X$ . Inmediatamente surge una imposición en costo de cómputo muy elevada: Dado que  $M_X \in \mathbb{R}^{784 \times 784}$  es simétrica, posee *rango completo* de autovectores. Calcular los 784 autovectores es pesado, y, aún provisto de ellos, multiplicar *todas* las imágenes contra la matriz generada también lo es. Esto es bloqueante, lo que buscábamos era *reducir* el problema en dimensión para alivianar el costo de cómputo.

Provisto de  $\alpha \in \mathbb{N}$ , y  $n_{iter} \in \mathbb{N}$ , buscamos generar la transformación  $P \in \mathbb{R}^{\alpha \times 784}$  tal que contenga  $\alpha$  **Componentes Principales** como filas. Como dichas componentes son los autovectores, buscamos  $\alpha$  autovalores y autovectores de la matriz de covarianzas  $M_X$ . Pero al tomar un número menor de componentes, se pierde información. Por eso mismo, decidimos **buscar las que maximicen la varianza**, son las que más información poseen en el espacio de información transformado. Recordando el fundamento del método planteado en la sección anterior, buscábamos los autovectores de  $M_X$  dado que la diagonalizaban. Al estar diagonalizada, los autovalores son los elementos de la diagonal, que a su vez son las varianzas de las variables en *nuevo espacio de datos*. Obtener los de mayor módulo nos aporta la mayor cantidad de información posible. Para esto aplica el **Método de la Potencia** (explicado en 2.3.1), iterando tantas

<sup>2</sup>En caso de empate, nos quedamos con el primero que hayamos encontrado que maximice la cantidad de apariciones

veces como el  $n_{iter}$  provisto, combinado con **Deflación** (desarrollado en 2.3.2). Repetimos  $\alpha$  veces un paso  $i$ , con  $B^0 = M_X$ , de: obtener  $\lambda_i$ , el  $i$ -ésimo autovalor ordenado por módulo, asociado a  $v_i$ , calcular  $B^{i+1} = B^i - \lambda_i v_i v_i^t$  y comenzar el paso  $i + 1$ .

Con lo cuál nos quedamos con una matriz  $P$  que posee, por filas, los  $\alpha$  autovectores de  $M_X$  que mayor información almacenan. El siguiente paso es aplicar el *cambio de base* a cada muestra  $z \in Z$  y a  $y$ :  $P y = \hat{y} \in \mathbb{R}^\alpha$  y  $\hat{z} = P z \in \mathbb{R}^\alpha$ , obteniendo sus correspondientes **Transformaciones Características**. A dicha transformación la denominamos  $tc_{PCA}$ .

### 2.1.3. Reducción de dimensión supervisada con PLS-DA

Presentamos un *pseudo-código* del procedimiento para luego explicar las decisiones involucradas y las condiciones correspondientes que se deben dar para su correctitud:

---

**Algorithm 1** PLS( $X, Y, \gamma$ )

---

```

1: for  $i \leftarrow [1..\gamma]$  do
2:    $M_i \leftarrow X^t Y Y^t X$ 
3:    $w_i \leftarrow$  autovector asociado al mayor autovalor de  $M_i$  {Debería estar normalizado, si no, normalizar}

4:    $t_i \leftarrow X w_i$ 
5:   Normalizar  $t_i$ 
6:    $X \leftarrow X - t_i t_i^t X$ 
7:    $Y \leftarrow Y - t_i t_i^t Y$ 
8: end for
9: return  $w_i$  para cada  $i \leftarrow [1..\gamma]$ 

```

---

El algoritmo recibe  $X \in R^{n \times 784}$ , la matriz de imágenes centralizadas por la media, e  $Y$  un vector que *mapea* cada posición con la etiqueta de la imagen que se encuentra en la susodicha posición en  $X$ . Dadas estas construcciones, buscamos ir obteniendo iterativamente los **autovectores dominantes**  $w_i$  (autovectores cuyos autovalores sean dominantes en el paso  $i$ ) sobre la matriz  $M_i$ . Notemos que  $M_i$  es simétrica en todos los pasos: siendo  $X = X_i$ ,  $Y = Y_i$  las matrices iniciales en el paso  $i$ , vemos que  $M_i^t = (X^t Y Y^t X)^t = (Y^t X)^t (X^t Y)^t = X^t (Y^t)^t Y^t (X^t)^t = X^t Y Y^t X = M_i$ . Como, por lo desarrollado en 1.4, requerimos  $w_i$  autovector dominante, utilizamos el **Método de la Potencia** (2.3.1) para extraerlo. El vector resultado se encuentra normalizado, por ende podemos evitar el paso de normalización siguiente a su obtención. Para finalizar la iteración, en base a  $w_i$ ,  $X_i$  e  $Y_i$ , calcular  $t_i$  como  $X w_i$  para realizar el cómputo de  $X_{i+1}$  e  $Y_{i+1}$ .

Las  $\gamma$  repeticiones de estos calculos nos otorga  $w_1, \dots, w_\gamma$  y los utilizamos para obtener la **Transformación Característica** de una imagen  $x_i$  como  $tc_{PLS}(x_i) = (w_1^t x_i, \dots, w_\gamma^t x_i) \in R_\gamma$ .

Con este planteo, tenemos un PLS tracional. La extensión a PLS-DA la hacemos tomando la  $Y$  como una matriz que centraliza, como a  $X$ , a una matriz que tiene un 1 en la posición  $(i, j)$  si la imagen  $i$  tiene etiqueta  $j^3$  o  $(-1)$  en caso contrario.

### 2.1.4. Clasificación inteligente: Transformación + kNN

Tanto PCA como PLS-DA nos facilitan una *transformación característica* que reduce la dimensión de las imágenes. Pero **por sí mismos no son clasificadores**. Que la dimensión de las imágenes se encuentre reducida, abre la posibilidad de utilizar el kNN y que su ejecución se complete en tiempos razonables.

Por lo tanto, el **algoritmo completo de clasificación**, sobre una imagen particular  $y \in Y$  a clasificar, se compone de tomar una  $tc_m$ , la transformación característica de PCA o PLS-DA, obtener  $tc_m(y)$  y  $tc_m(z_i)$ , con  $z_i \in Z$ , y finalmente utilizar el criterio de clasificación provisto por kNN.

---

<sup>3</sup>Indexando desde 1

## 2.2. Estrategias de medición

### 2.2.1. Evaluación robusta con *K-fold cross validation*

El objetivo de este trabajo es decidir una clasificación para un conjunto de entrada cuya categorización no es conocida. Dicho esto, es imposible medir la efectividad de los algoritmos planteados si no podemos determinar que la clasificación sea precisa. Por ende, decidimos partir en dos una base de entrada  $I$  cuya clasificación es conocida, para generar los conjuntos  $A$  y  $B$  (introducidos en 2.1.3). Decimos que las imágenes de  $A$  son las de **test** y las contenidas en  $B$  las de **train**. Como ventaja, esta decisión nos permite generar una transformación en base a las imágenes de *train*, correr los algoritmos clasificando las de *test* y comparar los resultados con las etiquetas reales para obtener métricas de calidad (2.2.2). Por otro lado, esta metodología correlaciona fuertemente los resultados a los conjuntos de entrada. Para ello decidimos implementar una técnica llamada ***K-fold cross validation***. A grandes rasgos, esta metodología propone **generar múltiples particiones** y ejecutar las rutinas sobre ellos. De esa forma, la elección de partición queda desacoplada del resultado y se puede hacer un análisis más objetivo de los datos obtenidos como resultado.

Para armar los conjuntos el criterio es el siguiente: según un parámetro  $K \in \mathbb{N}_{>0}$ , que indica el número de particiones distintas a generar, se toma  $\frac{1}{K}$  partes de  $I$  como *test* y el resto las consideramos de *train*. Por ejemplo, con  $K = 10$  extraemos el 10 % y se lo otorgamos a  $A$ , y el restante 90 % corresponde a  $B$ , con  $K = 5$  la misma idea pero con 20 % de *test* y el restante como *train*. Los conjuntos de la partición son llamados **folds**. Notar que uno tiene  $K$  maneras de tomar  $A$  y  $B$  de la manera que se menciona, por lo tanto, para validar la contundencia de la base utilizada se dice que: para cada  $i = 1, \dots, K$  uno **entrena** sobre los *folds* restantes que no sean  $i$  y luego **testea** (o valida) sobre el  $i$ .

La particularidad de la técnica es que trabaja sobre una *misma* base de datos. Por esa razón, **hay una imposición de responsabilidad fuerte para con la base ya que una poco representativa, demasiado pequeña o con baja uniformidad de los datos puede condicionar la experimentación**. Se decidió tomar como  $I$  los **dígitos manuscritos de la famosa MNIST database**<sup>4</sup>.

### 2.2.2. Métricas de calidad

Finalmente, se desarrollaron algunas métricas para medir qué tan buenas son las decisiones tomadas por el clasificador:

\* **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  $tp_i / (tp_i + fp_i)$ .

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 que tan bueno es un clasificador para, dada una clase particular, identificar correctamente a los pertenecientes a esa clase**. Dada una clase  $i$ , el recall de dicha clase es  $tp_i / (tp_i + fn_i)$ .

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 esta métrica para **medir un compromiso entre ambas**. Se define como  $2 * precision * recall / (precision + recall)$ .

<sup>4</sup>La cuál se puede [obtener gratuitamente](#) en el link



## 2.3. Algoritmos de Utilidad

### 2.3.1. Método de la Potencia

Tenemos una necesidad de encontrar *autovalores* con sus *autovectores asociados*. Para esto utilizamos el **Método de la Potencia**. Sea  $B^{n \times n}$  la matriz de entrada.

---

**Algorithm 2** Método de la Potencia( $B, x_0$ , condición de finalización)

---

```

1:  $v \leftarrow x_0$ 
2: while No se cumpla la condición de finalización do
3:    $v \leftarrow Bv$ 
4:   Normalizar  $v$ 
5: end while
6:  $\lambda \leftarrow v^t Bv$ 
7: return  $\lambda, v$ 

```

---

Este método busca un autovector  $v_1 \in \mathbb{R}^m$  tal que  $\|v\|_2 = 1$  aproximando el pasado como parámetro iterativamente, con la particularidad de que se corresponde con el autovalor  $\lambda_1 \in \mathbb{R}$  de manera que  $\lambda_1 > \lambda_i$  autovalor con  $i \neq 1$ . Las condiciones de convergencia son las siguientes:

- Los  $\lambda_1, \dots, \lambda_n \in \mathbb{R}$  autovalores de  $B$  deben cumplir que  $\lambda_1 > \lambda_2 \geq \dots \geq \lambda_n$ .
- Debe poseer  $\{v_1, \dots, v_n\}$  autovectores, con  $v_i$  asociado a  $\lambda_i$  con  $1 \leq i \leq n$ , que forman una base ortonormal. De esta forma  $v_1$  se denomina *autovector dominante*.
- El vector inicial  $x_0$  debe verificar que  $v_1 x_0^t \neq 0$ , en otras palabras no ser ortogonal al autovector dominante.

Notar que para dimensiones grandes, la probabilidad de elegir un vector no adecuado inicial al azar es prácticamente nula, de modo que el  $x_0$  es elegido en forma aleatoria. Las matrices a las que se les aplica el método, en este trabajo, no cumplen con *todas las precondiciones*. De todas formas, en la práctica, para matrices de tamaño grande y para un número elevado de extracciones, los autovalores son distintos, pero este es un argumento empírico. Utilizamos este algoritmo teniendo en mente las circunstancias.

### 2.3.2. Deflación

Este algoritmo soslayará un esquema iterativo en el cuál uno puede obtener autovalores y autovectores.

Sea  $B \in \mathbb{R}^{n \times n}$  una matriz que posee autovalores distintos  $\lambda_1, \dots, \lambda_n$  con autovectores  $v_1, \dots, v_n$  asociados tales que  $|\lambda_1| > \dots > |\lambda_n|$ , en otras palabras poder ordenarlos por módulo. Además se pide que los autovectores generen una base ortonormal. En la práctica, no es necesario que todas las desigualdades sean estrictas. Veamos entonces que:

$$\begin{aligned}
 (B - \lambda_1 v_1 v_1^t) v_1 &= Bv_1 - \lambda_1 v_1 (v_1^t v_1) = \lambda_1 v_1 - \lambda_1 v_1 = 0v_1 \\
 (B - \lambda_1 v_1 v_1^t) v_i &= Bv_i - \lambda_1 v_1 (v_1^t v_i) = \lambda_i v_i
 \end{aligned}$$

Por lo tanto, la matriz  $B - \lambda_1 v_1 v_1^t$  posee autovalores  $0, \lambda_2, \dots, \lambda_n$  asociados a  $v_1, \dots, v_n$  de tal forma que puedo repetir el proceso obteniendo  $\lambda_2$  y  $v_2$ . Este algoritmo se acopla muy bien con el Método de la potencia puesto que el susodicho extrae el autovalor dominante y su correspondiente autovector, con lo cuál es perfecto para una combinación de ambos.

### 3. Resultados y discusión

#### 3.1. Analisis Previo: Alfa y Gama

El uso de los métodos de PCA y PLS-DA se funda en reducir la dimensión de los datos a clasificar. La idea es capturar la mayor cantidad de información, dejando de lado las componentes que en determinado momento dejen de ofrecernos datos relevantes.

Para esto proponemos el siguiente método: Tomamos toda la base de entrenamiento, dejando vacía la base de *test*, y calculamos los 784 autovalores para ambos métodos. Sea  $\lambda_{PCA}$  la suma de los autovalores obtenidos por PCA y  $\lambda_{PLS-DA}$  la suma de los obtenidos por PLS-DA.

En el caso de PCA, como siempre calculamos los autovalores de la matriz de covarianzas, se cumple que la suma de los autovalores es igual a la suma de las varianzas, es decir, a traza de la matriz de covarianzas. Entonces podemos decir que  $\lambda_{PCA}^i / \lambda_{PCA}$  representa el porcentaje de varianza que absorbe el  $i$ -ésimo autovalor.

Si acumulamos las varianzas que absorbe cada vector, se genera el siguiente gráfico:

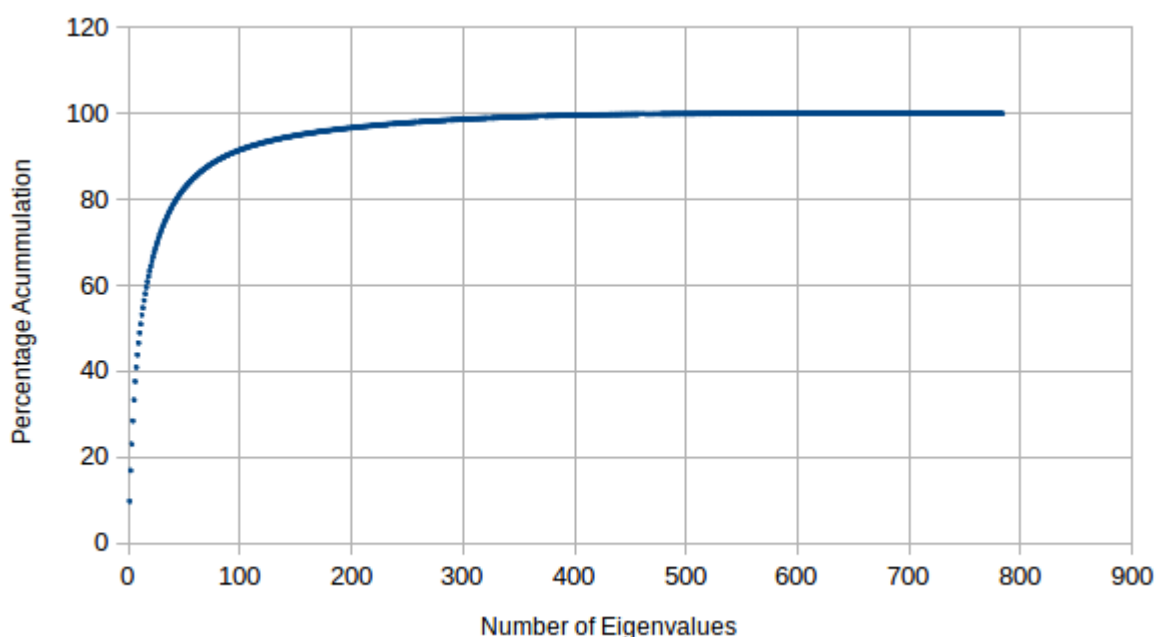


Figura 1: Porcentaje de varianzas acumuladas: PCA

Puede verse que el primer autovector se lleva el 10 % de la varianza, el segundo casi un 7 %. Pero es más interesante cuando se llega a tomar 50 autovalores, en este caso, ya se tomó cerca de 83 % de la información. Con lo cual, podríamos pensar los demás autovalores no son tan significativos <sup>5</sup>.

Para el caso de PLS-DA, no se calculan los autovalores de la misma matriz en cada paso, haciendo que el análisis anterior no sea del todo directo. Notemos que el autovalor obtenido en cada paso, si bien es de una matriz diferente, tiene la idea de maximizar las varianzas y minimizar las covarianzas de la unión entre las imágenes y sus etiquetas. Entonces decidimos aplicar el mismo procedimiento, obteniendo estos resultados:

<sup>5</sup>El 90 % se alcanza con 86 autovalores. El 99 % con 330

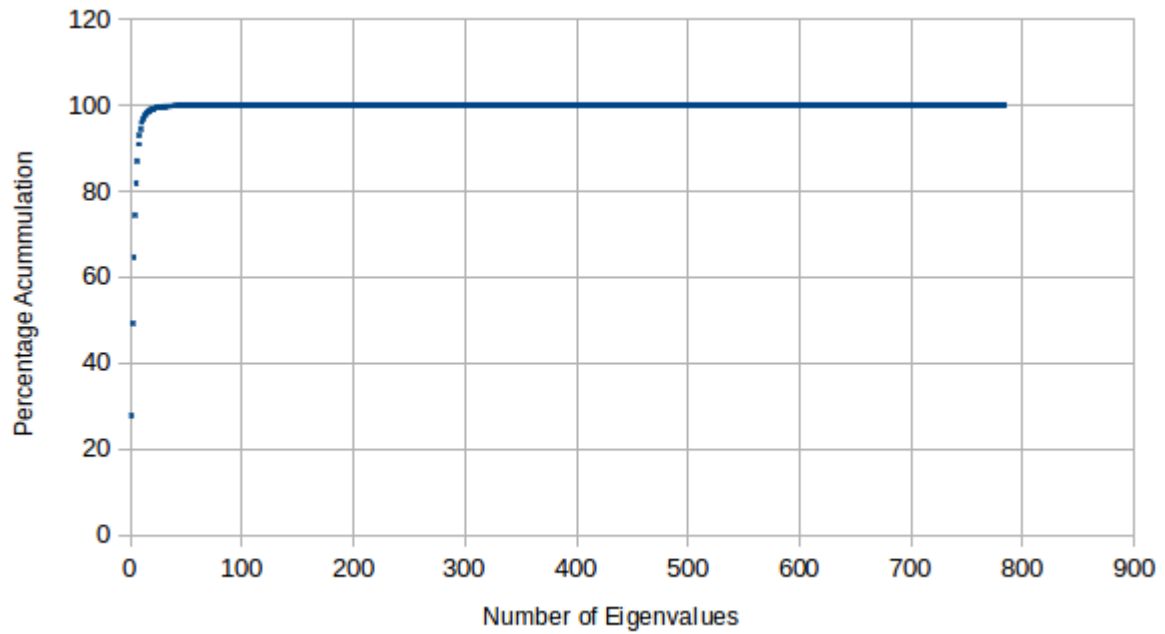


Figura 2: Porcentaje de varianzas acumuladas: PLS-DA

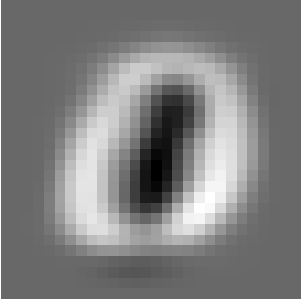
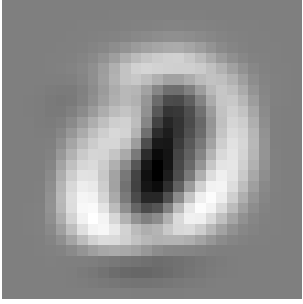
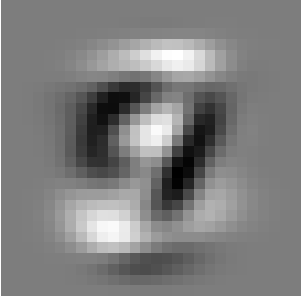
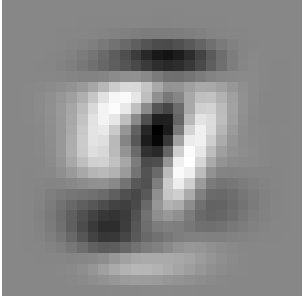
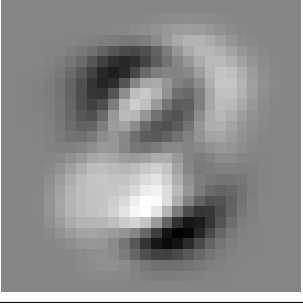
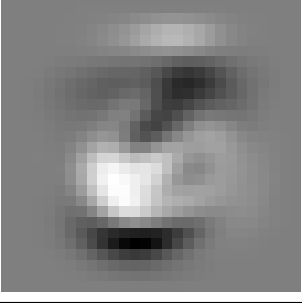
En este se ve que ya el primer autovector, captura cerca del 30 % de la información. Junto al segundo, consumen casi el 50 %. Esto indica que tener en cuenta la etiqueta a la hora de intentar reducir la dimensión de los datos puede ser muy útil. De hecho, con 8 autovalores se captura el 90 % y con 18 ya el 99 %

Los subsiguientes análisis cualitativos fueron realizados bajo estas conclusiones. Es decir, no consideramos tomar  $\alpha$  mayor a 300 ni  $\gamma$  mayor a 20.

### 3.2. El nuevo espacio

Cada autovector que buscamos, proporciona nueva información para nuestro análisis. En esta sección vamos a intentar ver cuál es esta información, es decir, qué es lo que nos está diciendo cada autovector.

Vamos a tomar únicamente los primeros 3 autovectores obtenidos por cada método, porque no se pueden graficar puntos en más de 3 dimensiones.

#	PCA	PLS-DA
Primer Autodígito		
Segundo Autodígito		
Tercer Autodígito		

Cuadro 1: Primeros autodígitos de los métodos

La tabla anterior muestra los 3 primeros autodígitos obtenidos para cada método. Para obtener cada uno, se obtuvo el autovector asociado a cada uno de los primeros 3 autovalores, que están normalizados. Se los convierte a base 0 - 1<sup>6</sup> y se los multiplica por 255. Así obtenemos una matriz con valores entre 0 y 255 que representa una imagen, en particular la de los autodígitos.

**Hipótesis:** Dada la forma del primer autodígito, el primer autovector no admite confusión entre los dígitos 0 y 1.

Para ver si esto vale, vamos a mostrar el extracto pertinente de las matrices de confusión obtenidas para PCA y PLS-DA respectivamente:

Actual\Predicted	0	1
0	3009	0
1	0	4091

Actual\Predicted	0	1
0	3213	0
1	0	3924

Se puede apreciar que la confusión entre ambos dígitos es nula, el resto de la matriz no aporta grandes resultados, a excepción que para el dígito 1:

Para PCA:

<sup>6</sup> $(x^i - x_{min}) / (x_{max} - x_{min})$ , con  $x_{min}$  el mínimo valor encontrado en  $x$  y  $x_{max}$  el máximo,  $x^i$  el  $i$ -ésimo elemento de  $x$

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
1	0	3649	57	44	151	45	61	374	65	238

Para PLS-DA

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
1	0	3924	8	25	91	7	7	421	24	177

Muestran que cuando la imagen a etiquetar es un 1, en la mayoría de los casos va a acertar, pero si no lo hace, es más probable que lo confunda con un 7 o con un 9, lo cual tiene sentido ya que los dígitos se parecen en su forma.

Es interesante entonces mostrar como se distribuyen los puntos en el nuevo espacio <sup>7</sup>.

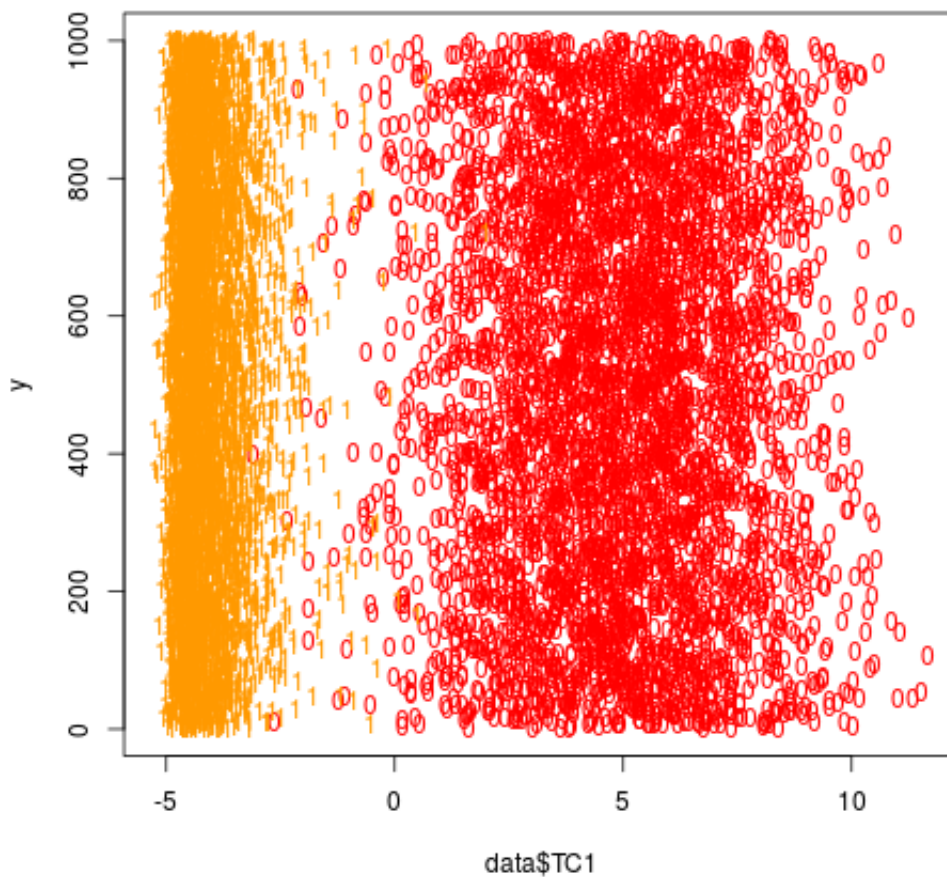


Figura 3: 0 VS 1. Puntos en el nuevo espacio PCA

<sup>7</sup>El eje  $y$  no es representativo, se tomo para simular  $\mathbb{R}^2$

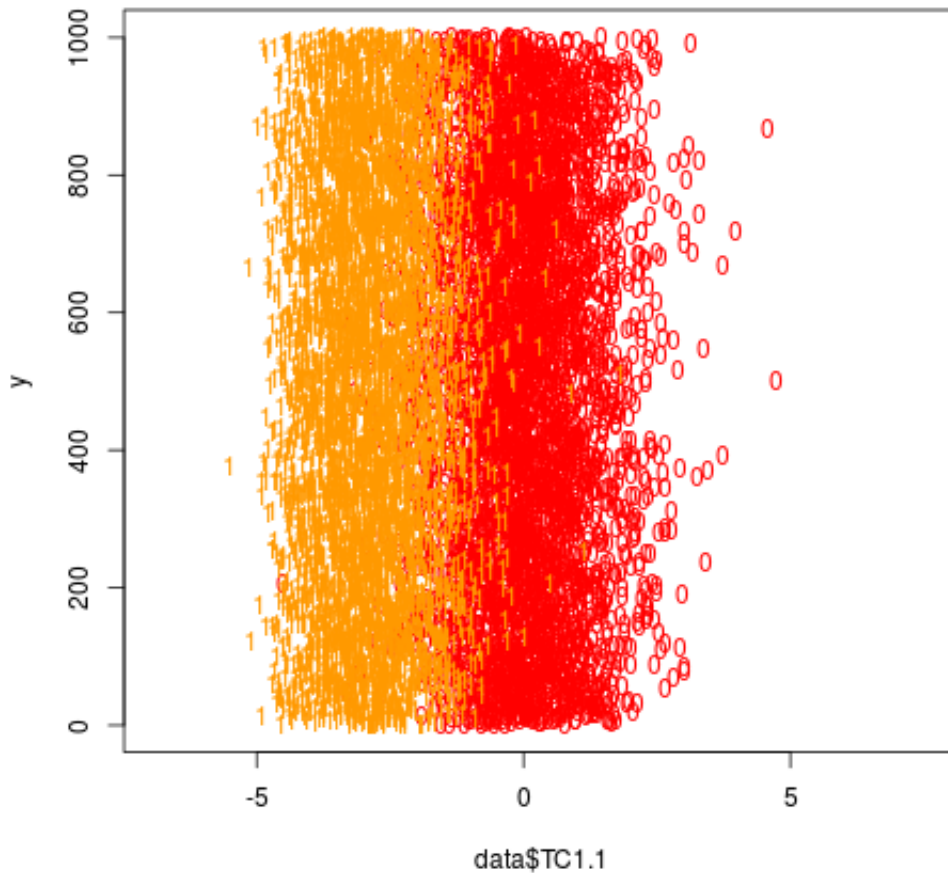


Figura 4: 0 VS 1. Puntos en el nuevo espacio PLS-DA

En PCA se ve una clara distancia entre los puntos transformados con etiquetas 0 y 1, guiados por el rasgo distintivo del autodígito. En PLS-DA parecería que se confunden en la frontera, aunque la mayor densidad de puntos está en la media de cada uno, por lo que kNN clasificando un punto sobre la frontera no debería tener problemas <sup>8</sup>.

Si el problema fuera clasificar 0's de 1's, con el primer autovector debería alcanzar, pero como pretendemos extender esto a clasificar todos los dígitos, buscamos el segundo, que para ambos casos tiene forma de 9.

Como antes, mostramos las matrices de confusión.

<sup>8</sup>No pudimos encontrar una explicación para este fenómeno, creemos que la mayoría de los 1's que se acercan a la media del 0 están en la base de entrenamiento

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	3080	1	282	71	7	175	414	4	92	6
1	0	4353	66	46	15	54	19	33	85	13
2	448	102	1017	792	92	538	494	48	609	37
3	117	90	532	2098	60	437	293	39	660	25
4	8	57	40	17	1704	91	216	820	43	1076
5	359	78	527	609	181	701	633	62	560	85
6	516	76	460	253	204	531	1299	58	661	79
7	1	131	29	39	1109	98	103	1842	62	987
8	273	109	553	703	121	568	653	41	984	58
9	34	83	34	13	1383	92	154	1246	53	1096

Cuadro 3: Confusion Matrix PCA

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	3247	1	146	45	13	190	442	2	42	4
1	0	4429	63	46	8	23	13	17	79	6
2	247	127	1167	802	82	529	540	48	594	41
3	60	89	569	2005	51	422	346	53	714	42
4	7	34	29	33	1754	100	140	781	70	1124
5	285	45	508	546	180	732	828	47	581	43
6	499	64	460	364	155	694	1167	34	665	35
7	4	90	23	36	1030	63	60	1949	70	1076
8	184	120	504	763	170	553	676	58	974	61
9	26	32	29	31	1346	75	102	1288	53	1206

Cuadro 4: Confusion Matrix PLS-DA

Como primera impresión de ambas tablas notamos que no se confunde 0 con 1 ni 1 con 0 salvo en un caso que debe ser un punto aislado en la muestra.

Lo destacable aquí es que los 4, 7 y 9 o bien acierta o bien los confunde entre sí, salvo un pequeño porcentaje de la muestra (razonable pues aún no tenemos mucha información). Estos tres números tienen la particularidad de un “palo” que los distingue de los otros, el 1 también lo tiene, pero este suele escribirse como un “palo” mientras que los otros tres son un “palo” más “algún garabato” extra.

De esto podemos decir que el segundo autodígito distingue entre 4, 7 y 9 de los demás.

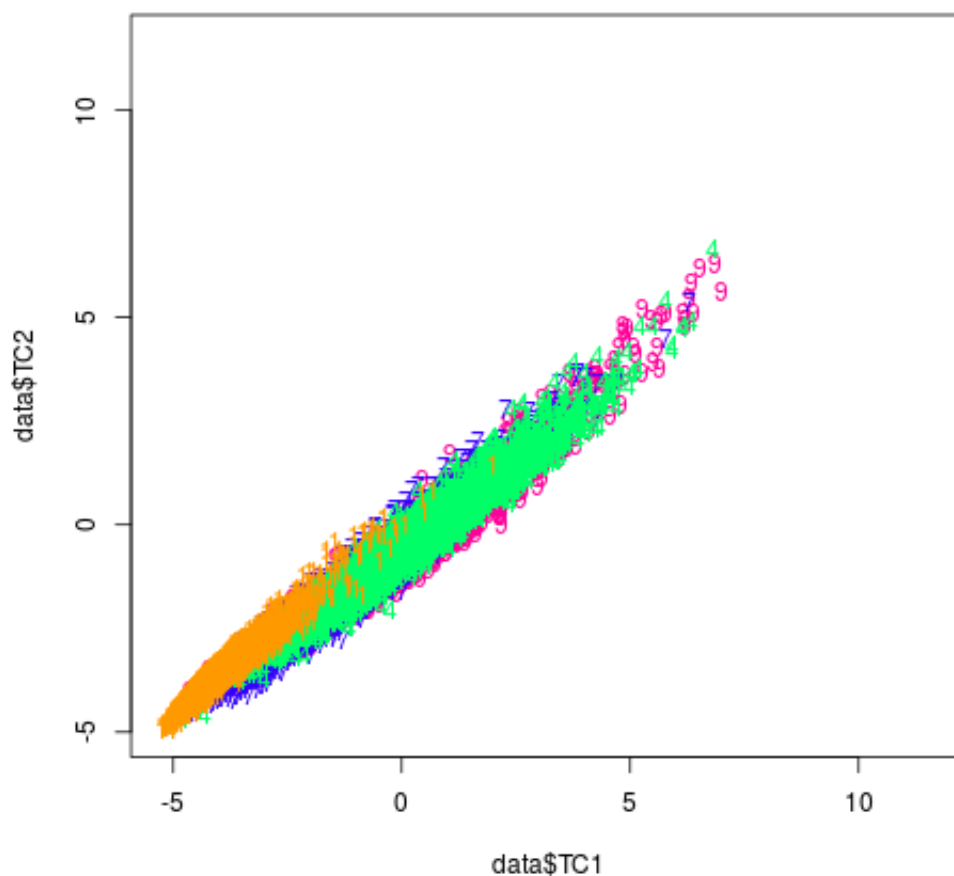


Figura 5: 1 VS 4 VS 7 VS 9. Puntos en el nuevo espacio PCA

Este es el espacio al que llegan los 4, 7 y 9<sup>9</sup>, casi que se superponen entre sí. Mientras el 1 está superpuesto de alguna manera, pero en un sector más chico, donde es más denso y por ende, predominante.

En el siguiente gráfico, se ve que la superposición para PLS-DA no se da tanto para el 1, lo cual se refleja en la matriz de confusión que indica que hay menos confusiones de 4, 7 y/o 9 con el 1 que con PCA.

Esto además trae a la luz que ahora es más difícil confundir al 1 con los demás, pues con los que más competía eran el 4, 7 y 9 y ahora estos fueron aislados a otro lugar del espacio.

<sup>9</sup>el eje  $y$  son los puntos que genera el segundo autovector



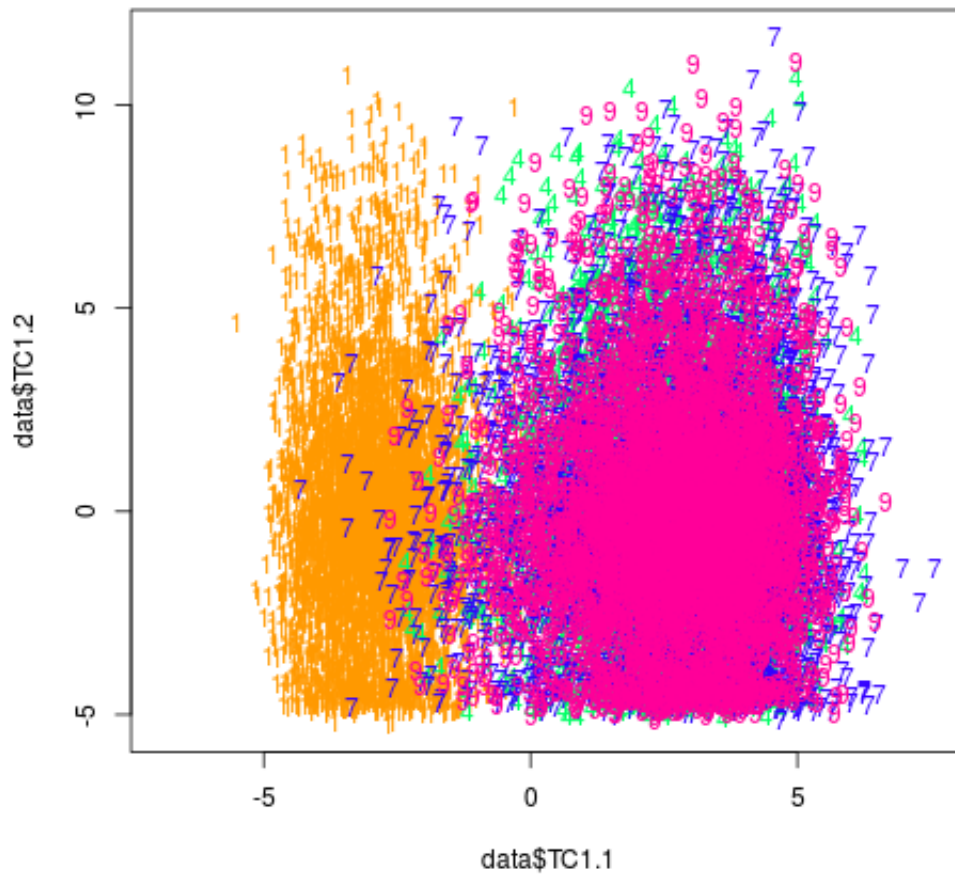


Figura 6: 1 VS 4 VS 7 VS 9. Puntos en el nuevo espacio PLS-DA

Finalmente analizaremos el tercer autódígito, en este caso son distintos para los dos métodos. El de PCA se parece a un 3, mientras que PLS-DA se parece a un 6.

En este caso no mostraremos las matrices de confusión, pero destacaremos que se siguen manteniendo las dos características previas. Además notamos que la cantidad de aciertos sobre el total de casos de test es notablemente mayor para PLS-DA (66 % contra 50 %). Esto viene asociado con lo analizado en la sección 3.1, pues PCA hasta el momento capturó el 23 % de la información mientras que PLS-DA el 65 %.

Ahora mostraremos la distribución en este espacio de los puntos. Para mayores referencias:

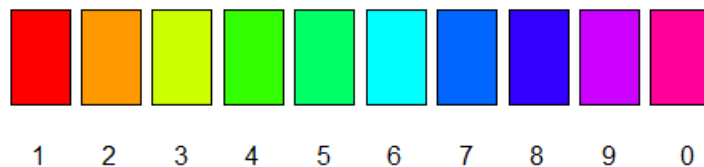
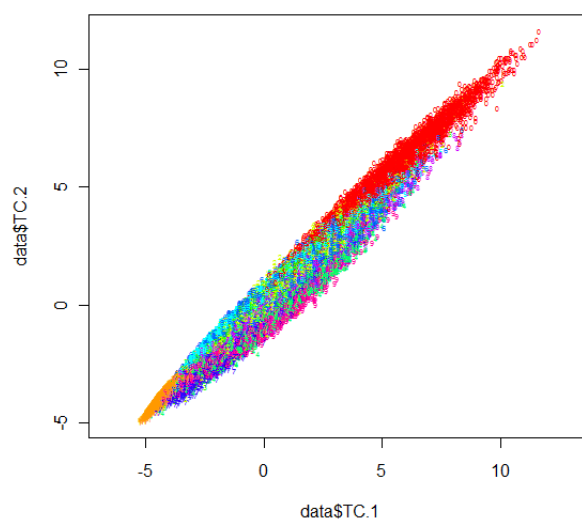
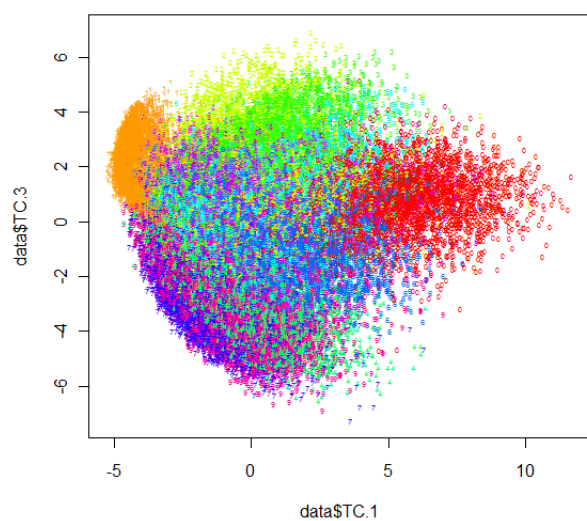
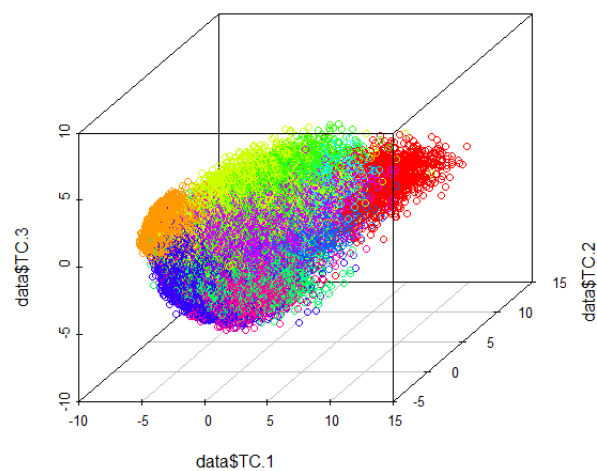
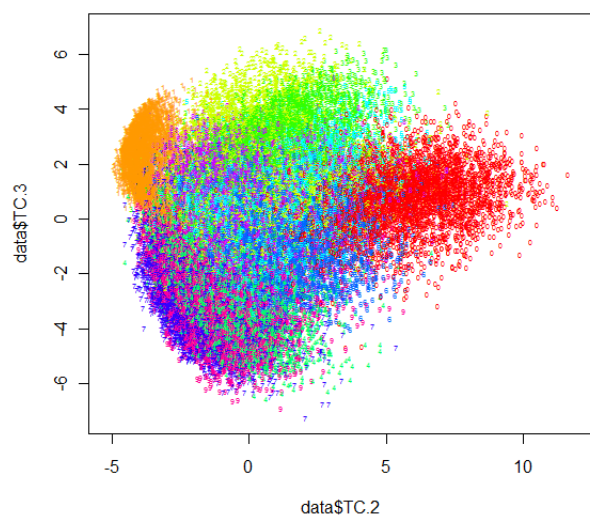
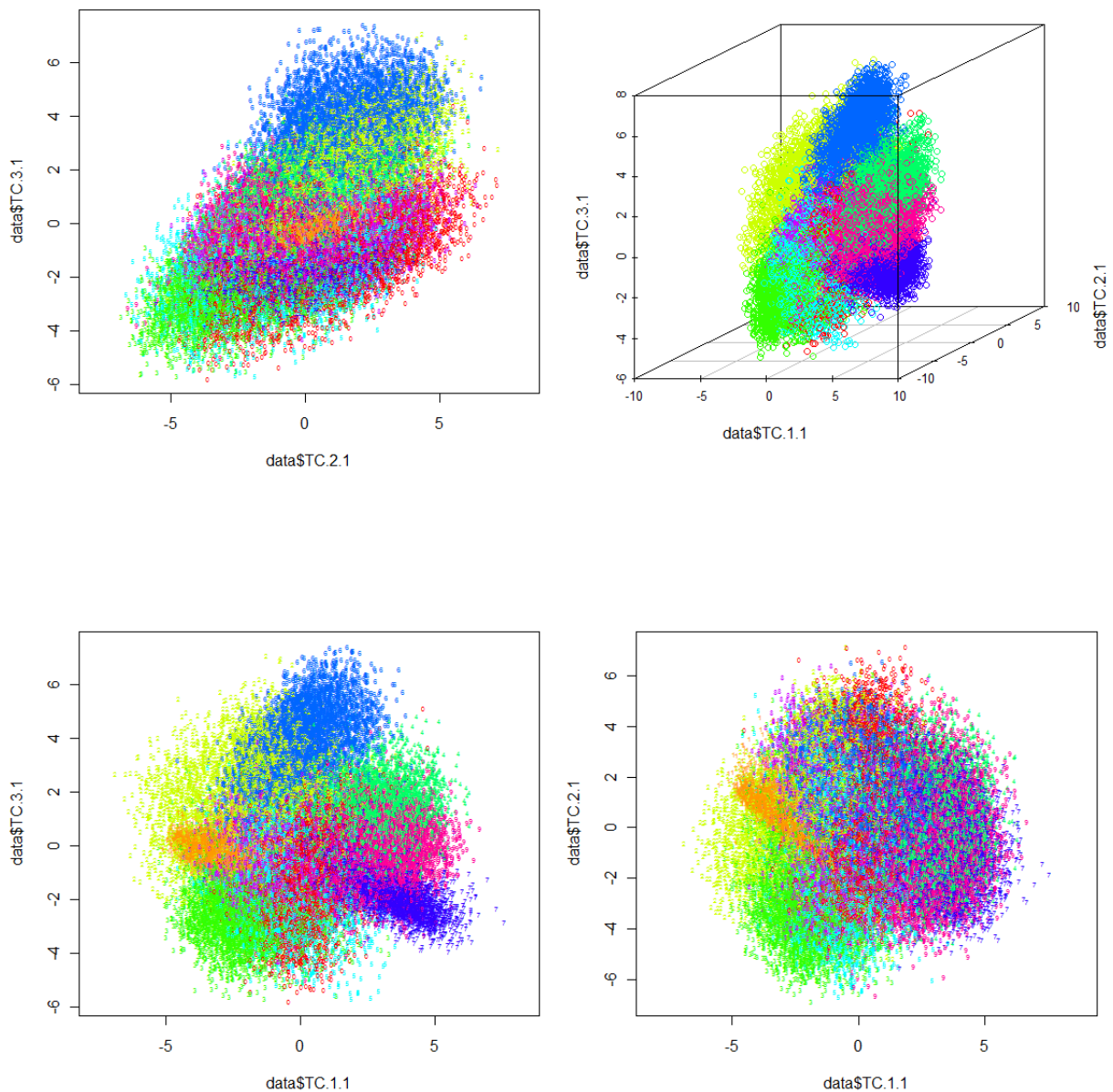


Figura 7: Referencias de colores

Los siguientes 8 gráficos, corresponden, los primeros 4, a la distribución de los puntos según PCA y los últimos 4, a la distribución según PLS-DA.





En ambos se ve como se empiezan a formar burbujas donde se concentran los distintos dígitos. Como se dijo anteriormente, en PLS-DA es más notorio. Los gráficos que no son de las 3 dimensiones, corresponden a las proyecciones sobre alguno de los ejes, donde se ven algunas burbujas solapadas, pero que el gráfico 3D nos muestra que comienzan a separarse en los distintos planos.

Si bien no esperamos que las burbujas se separen por completo (tener un 100 % efectividad), sí esperamos que a mayor cantidad de dimensiones consideradas, alcancemos un hit-rate lo suficientemente alto en contraste al tiempo de ejecución de los algoritmos.

### 3.3. Análisis de KNN

#### 3.3.1. Costo temporal

El algoritmo KNN es temporalmente costoso ya que para calcular la distancia entre dos vectores hay que considerar a todas sus coordenadas. Por lo tanto somos muy dependientes del tamaño del vector, que en general al trabajar con imágenes suelen ser bastante grandes.

Para analizar nuestro dataset con KNN utilizamos cross validation con  $K = 5$  y  $K = 10$ .

En la siguiente figura se pueden ver los tiempos de ejecución promedio de las particiones para cada valor de  $K$ .

KNN Image Process Duration		
	5 Partitions set	10 Partitions set
KNN Total Time Avg	6m 45s	4m
KNN Total Time STDev	10s	4s
KNN Per Image Time Avg	48ms	57ms
KNN Per Image Time STDev	1ms	1ms

Figura 8: KNN Image Process Duration

Para el caso de  $K = 5$  obtuvimos que etiquetar una imagen usando KNN tardó en promedio 48ms, que aunque parece poco, al correr todo nuestro set de test de 8400 imágenes se obtuvo un tiempo de ejecución promedio de 6.75 minutos con 10 segundos de desvío estándar. Se puede observar fácilmente que si quisiésemos clasificar un millón de imágenes se tardaría aproximadamente 13 horas.

Por otro lado, vemos que si realizamos 10 particiones a nuestro dataset, el tiempo de procesamiento de cada imagen aumenta de 48ms a 57ms. Esto se debe a que con 10 particiones se debe buscar el vecino más cercano entre más imágenes.

Debemos aclarar también que el costo temporal de variar la cantidad de vecinos de KNN es lineal y por lo tanto esto no tiene una influencia significativa en el tiempo de ejecución. En consecuencia decidimos analizar la calidad de los resultados de KNN disminuyendo la importancia del costo de agregar más vecinos.

#### 3.3.2. Calidad del algoritmo

Analizamos la calidad de KNN (con  $K = 5$ ) calculando su Hit Rate, Precision, Recall y F1 score. Se pueden observar los resultados en el siguiente gráfico.

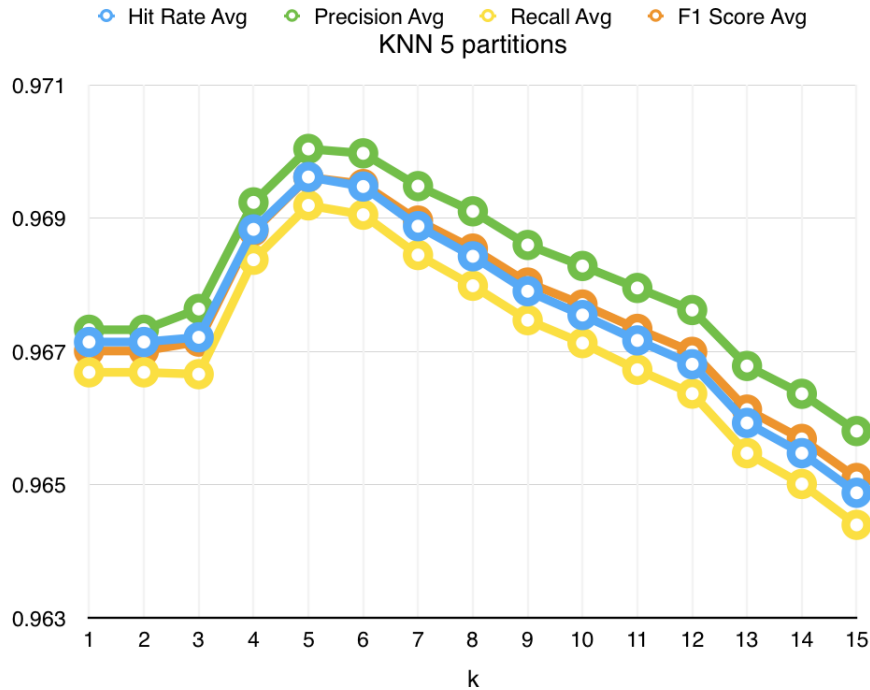


Figura 9: KNN quality comparison

Una de las primeras cosas que se pueden observar es que la calidad de los resultados es extremadamente buena. El hit rate promedio para el mejor  $k$  es 0.97. Es notable el valor y la similitud entre los resultados de las distintas métricas. Ese rasgo denota una *matriz de confusión* subyacente que es saludable. Dicho fenómeno se debe a que:

- Por una parte, la *precision* nos indica cuántos *aciertos reales* hubo dentro de los *predichos*. (ej: porcentaje de 8's predichos correctamente dentro de todos los clasificados como 8)
- Por la otra, *recall* otorga el porcentaje de *aciertos* dentro de todos los elementos pertenecientes a una clase. (ej: porcentaje 8's predichos dentro de todos los 8's reales)

Al haber obtenido buenos y similares resultados en las susodichas métricas, se habilita un análisis más objetivo del *hit-rate* puesto que suele aplanar síntomas negativos ya que es meramente “adivinados sobre totales”. Esta relación entre ambas métricas es resumida en el *F1-Score* y natural que sus valores sean elevados ya que es una relación entre ambos. Es cierto que hay una preponderancia de *precision* por sobre *recall*, con lo cuál uno estaría tentado a decir que es mayormente contundente dentro de sus propias predicciones que en lo que en realidad ocurre. Esta afirmación es empíricamente cierta, pero de todas formas la diferencia de  $\sim 0.005$  que presentan no creemos que es suficiente como para ser concluyentes.

Por otro lado también se puede ver que, a partir de  $k = 5$ , al aumentar la cantidad de vecinos la calidad de los resultados disminuye. Esto se debe probablemente a que considerar más imágenes comienza a agregar ruido a la medición. Al considerar más vecinos que cada vez se empiezan a parecer menos, es posible que se incorporen un conjunto cada vez mayor de imágenes que pertenecen a otra clase. Así mismo podemos ver que desde 1 vecino hasta 5 la información que aporta agregar más vecinos es valiosa y ayuda mejorar la clasificación.

Analizando el hit rate más detalladamente obtenemos el siguiente gráfico donde podemos observar su desvío estándar.

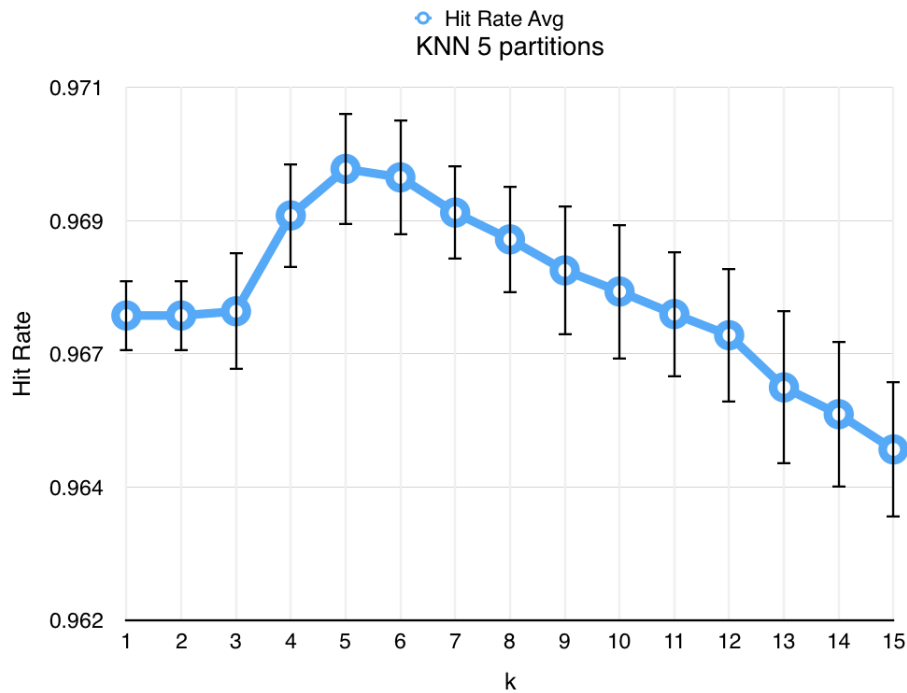


Figura 10: KNN hit rate

Por un lado podemos ver que el desvío estándar es relativamente pequeño. Este desvío representa los diferentes resultados del Hit Rate para cada partición de nuestro data set. Entonces sabemos que todas nuestras particiones tienen un buen resultado y no estamos sesgando nuestro testeó. Otra cosa interesante que se desprende del gráfico es que los desvíos estándar van aumentando a medida que se incrementa la cantidad de vecinos de KNN. De hecho en el siguiente gráfico podemos ver este incremento.

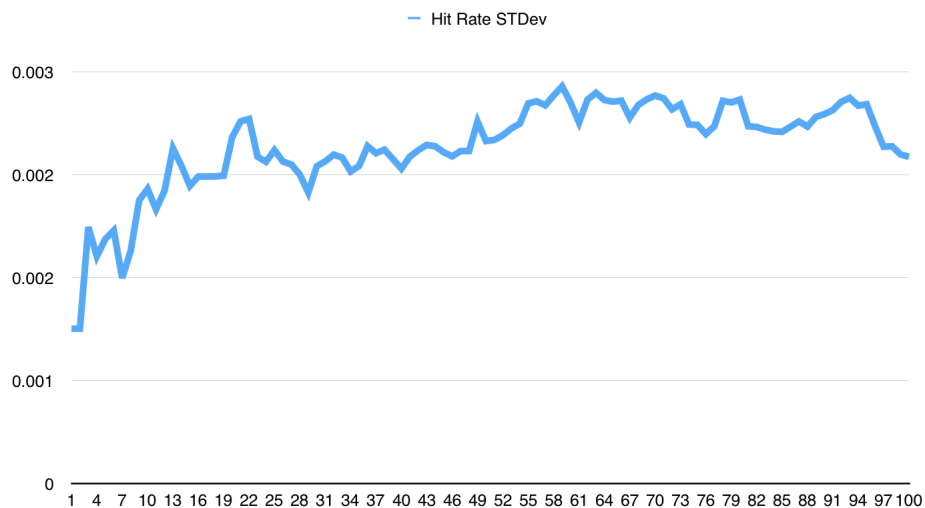


Figura 11: KNN hit rate standard deviation

Aunque los números se mantienen pequeños, es claro que agregar más vecinos incrementa el ruido y hace divergir más la calidad de los resultados en cada partición.

Una vez determinado que  $k = 5$  optimiza la calidad de KNN y además no compromete el tiempo de

ejecución, veamos en particular cómo está clasificando a nuestro dataset.

**KNN 5 partitions Confusion Matrix**

		Predicted									
Actual		0	1	2	3	4	5	6	7	8	9
	0	4107	1	2	0	1	4	12	1	1	3
	1	0	4656	7	2	4	1	2	7	1	4
	2	22	32	4013	14	3	3	6	71	8	5
	3	5	10	20	4191	0	48	4	25	26	22
	4	2	42	0	0	3927	0	12	4	0	85
	5	11	3	1	42	2	3657	45	3	8	23
	6	15	4	1	1	5	18	4092	0	1	0
	7	1	46	13	1	5	0	0	4287	1	47
	8	12	47	14	54	13	67	16	13	3785	42
	9	15	8	4	28	44	10	2	62	6	4009

Figura 12: KNN Confusion Matrix

Esta matriz de confusión es la suma de las matrices de confusión de cada una de las 5 particiones del test. Nos muestra claramente en su diagonal las clasificaciones acertadas y en el resto de las posiciones las que no clasificó correctamente. Vemos que los números fuera de la diagonal en general son bajos y que los más altos en general son casos que justifican cierto error. Por ejemplo, hubo setenta y un 2 que clasificó como 7 y a su vez hubo cuarenta y seis 7 que clasificó como 2. También vemos cierta confusión entre el 4 y 9, 8 y 5, etc. Se puede ver claramente como la matriz de confusión se condice con los datos plasmados en el gráfico de las métricas. Esto corrobora que de la diferencia entre *precision* y *recall* no denotaba una inclinación relevante por una o la otra, por ejemplo, con los dígitos 1 y 2: la *precision* para la clase 1 es notoriamente menor que para 2, y lo contrario ocurre con *recall*. El resto tiene mayor equitatividad entre ambas métricas si uno observa con detenimiento.

Todo este análisis realizado para  $K = 5$  lo hicimos también para  $K = 10$ .

En el siguiente gráfico podemos observar que las medidas de calidad son similares al anterior caso y  $k = 5$  sigue siendo el máximo.



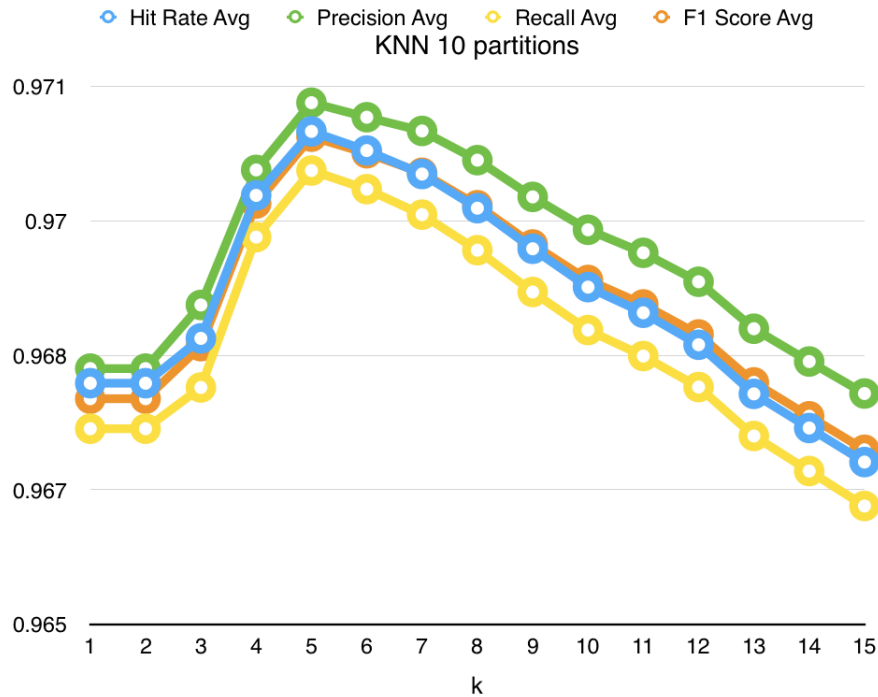


Figura 13: KNN Hit Rate (10 partitions)

Comparando los desvíos estándar se puede ver que con 10 particiones el desvío es bastante superior al caso anterior. Esto probablemente se deba a que al haber más particiones de tamaño más pequeño, los casos que la base de entrenamiento no sabe reconocer no estén tan uniformemente distribuidos en esos 10 conjuntos. Un análisis *cuasi* idéntico de comparación de métricas hecho con  $K = 5$  puede aplicarse para este caso también. Lo remarcable es el nivel de consistencia que se mantiene entre los 2 experimentos, lo cual es un argumento a favor en la calidad de las metodologías de medición planteadas.

Por último comparamos el hit rate entre los 2 conjuntos de particiones y observamos que el caso de 10 particiones tiene un hit rate ligeramente superior. Esto se debe a que su base de entrenamiento es más grande y puede así reconocer mejor los dígitos.



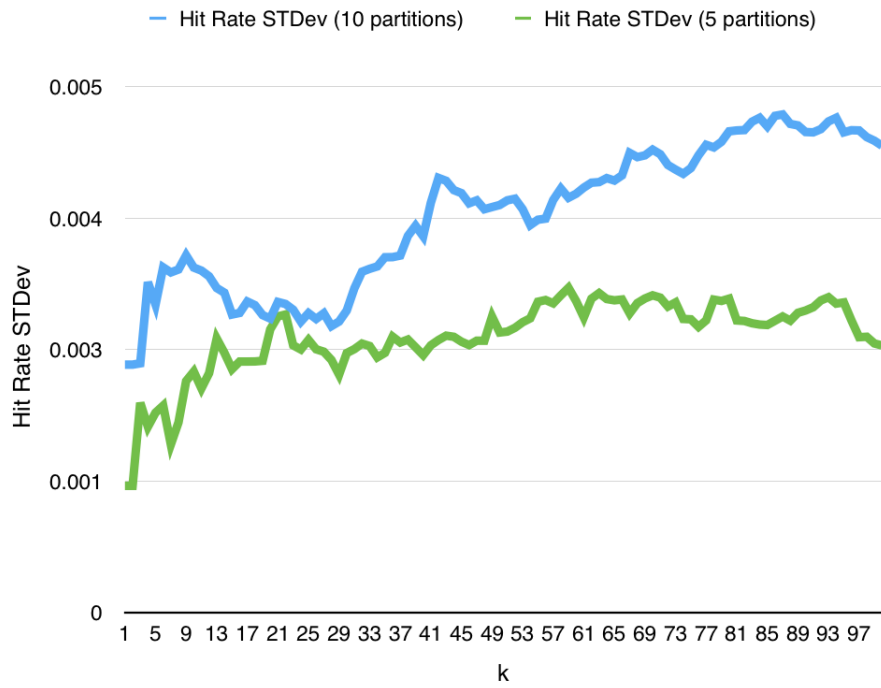


Figura 14: KNN Hit Rate (10 partitions)

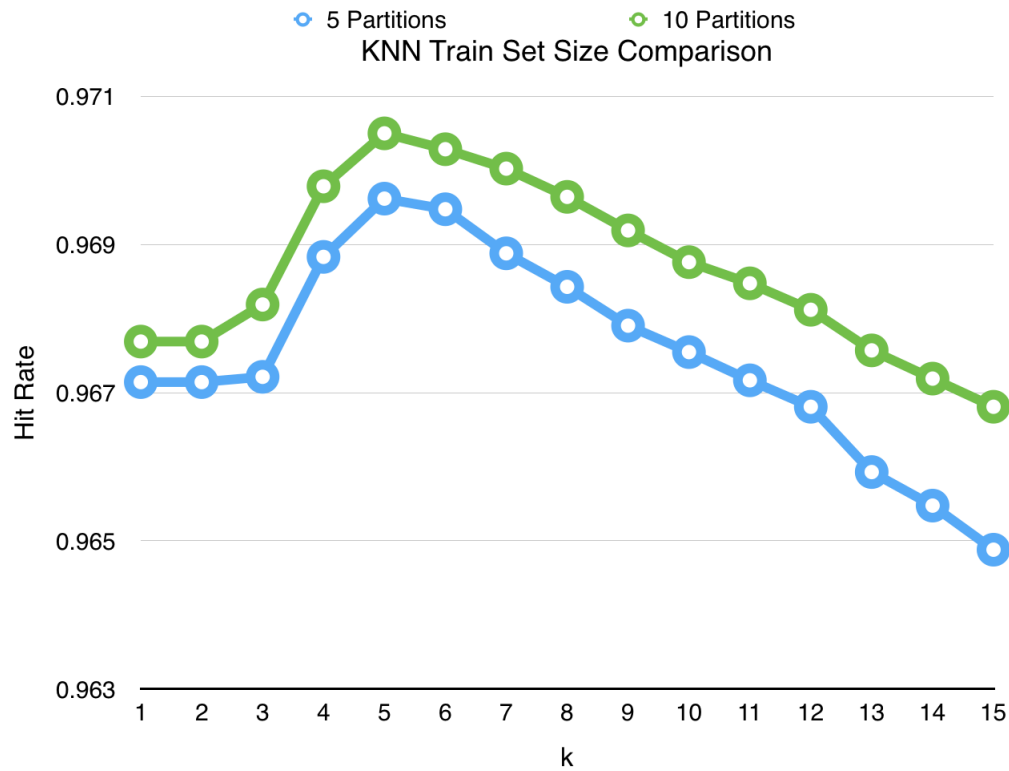


Figura 15: KNN Hit Rate (10 partitions)

### 3.4. Análisis de PCA y PLS-DA con kNN

#### 3.4.1. Descripción

En esta sección trataremos de encontrar cuáles son los parámetros que optimizan los resultados de KNN al preprocesar la información usando PCA y PLS-DA. Para esto tendremos en cuenta los tiempos de ejecución de los algoritmos para encontrar la mejor relación entre calidad y costo temporal.

#### 3.4.2. PCA

##### Costo temporal

Al igual que en el experimento anterior, la variación de  $k$  no altera significativamente el costo temporal de KNN. Por lo tanto para analizarlo, nos preocuparemos de calcular únicamente el rango ideal de valores para el parámetro  $\alpha$ . Los siguientes gráficos muestran exactamente eso.

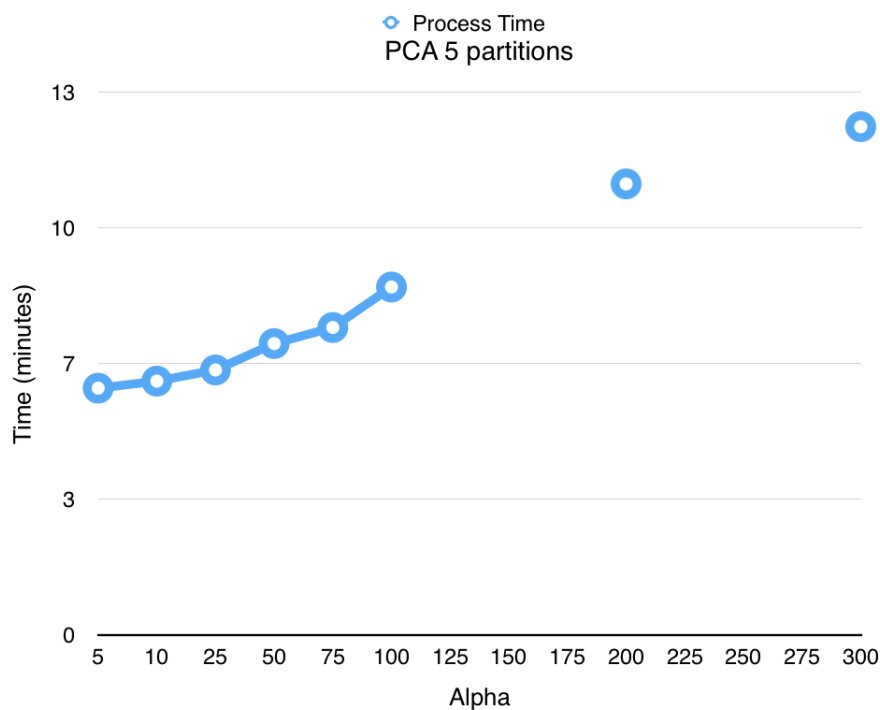


Figura 16: PCA data set training

En este gráfico podemos ver el tiempo que lleva el entrenamiento de PCA para reducir la dimensión de los vectores. Para algunos valores intermedios no fue calculado este valor debido al tiempo de cómputo que se requería, pero de todos modos podemos percibir una complejidad aproximadamente lineal, al menos en el rango que nos compete.

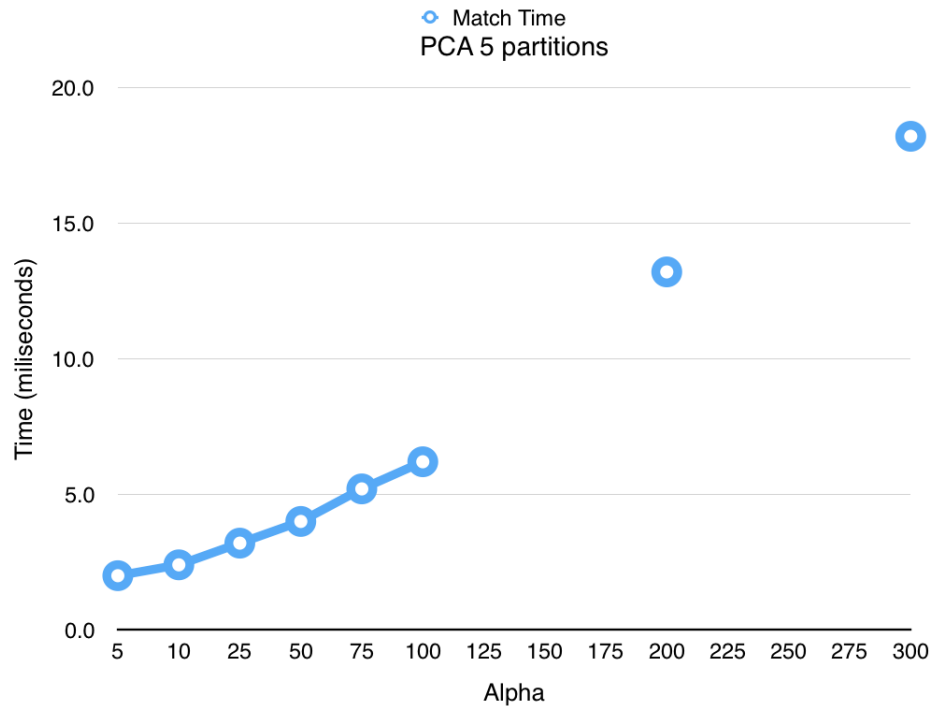
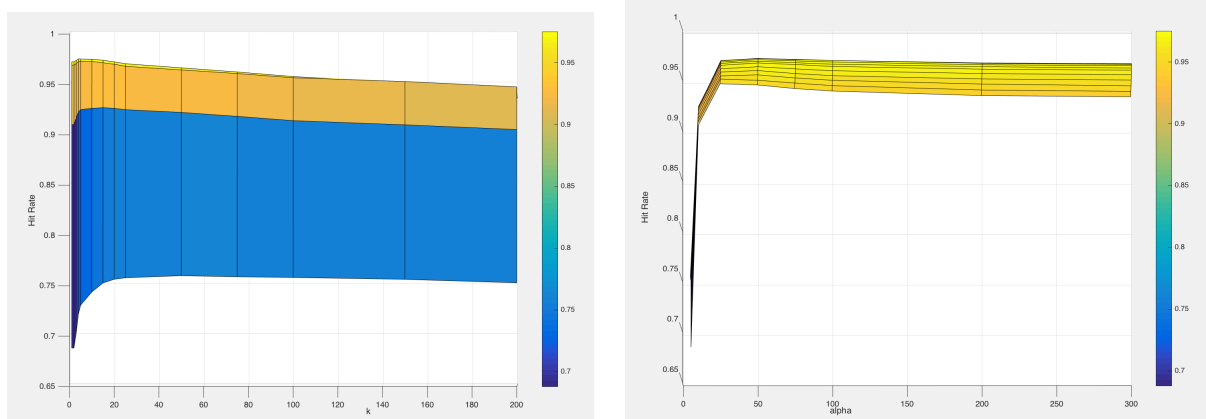


Figura 17: KNN Image Process Duration After PCA

En este otro gráfico se aprecia el tiempo en milisegundos que tarda una nueva imagen en ser clasificada usando KNN + PCA, una vez entrenado el data set. En este caso también podemos notar la linealidad de la función.

Por lo tanto podemos concluir que en los rangos adecuados, la complejidad temporal tanto del entrenamiento como de la clasificación es lineal respecto a  $\alpha$ . Por lo tanto emprendemos el análisis de la calidad de los resultados teniendo este factor en cuenta.

### Calidad del algoritmo

Cuadro 9: Hit Rate para valores de  $\alpha$  y  $k$ 

En estas figuras graficamos los valores del hit rate para valores de  $\alpha$  entre 5 y 300, y de  $k$  entre 1 y 200. Las primeras 2 figuras nos muestran los perfiles.

En la primera se puede observar la curva para los valores de  $k$ . Estos tienen un pico en  $k = 4$  y luego

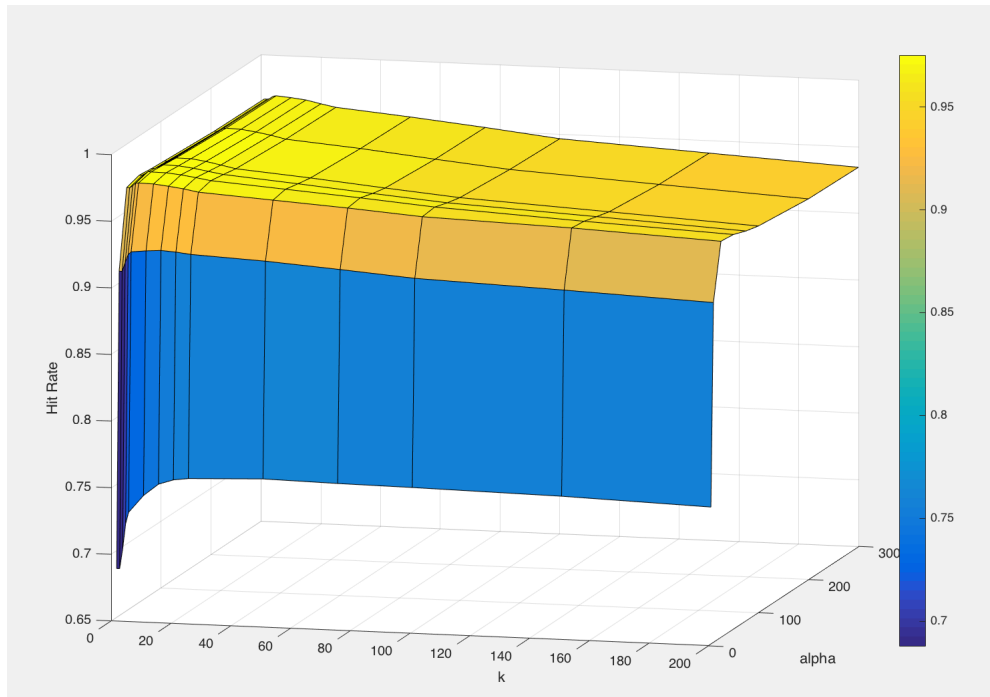


Figura 18: PCA data set training

comienza a descender. Esto al igual que en KNN puro se debe probablemente a que agregar muchos vecinos empieza a generar ruido en la clasificación de la imagen. Vemos que pocos vecinos tampoco generan el resultado óptimo.

Por otro lado en la segunda imagen vemos cómo fluctúa el hit rate respecto a  $\alpha$ . En este caso es interesante ver que menos de 10 autovectores dan un resultado bastante pobre y que encontramos un máximo en  $\alpha = 50$ . A partir de ahí el hit rate comienza a descender. Esto es contraintuitivo ya que en teoría al aumentar el  $\alpha$  estamos dando más información. Es probable que la información que comienzan a dar los nuevos autovectores ensucien más de lo que ayudan.

En el último gráfico tenemos una vista más general de ambos parámetros. Aquí se puede ver que tenemos nuestro máximo en  $\alpha = 50$  y  $k = 4$ . Lo interesante es que no hay que comprometer demasiado el tiempo de cómputo con ese  $\alpha$ .

### 3.4.3. PLS-DA

#### Costo temporal

Al igual que para PCA, nos centramos en analizar valores de  $\gamma$  únicamente para determinar un rango adecuado de valores a considerar. Las siguientes imágenes muestran los resultados obtenidos.

En esta imagen se puede apreciar también cierta linealidad en el costo temporal del entrenamiento de PLS. Vemos incluso que el desvío estándar es bastante pequeño entre las particiones así que el gráfico es bastante representativo.

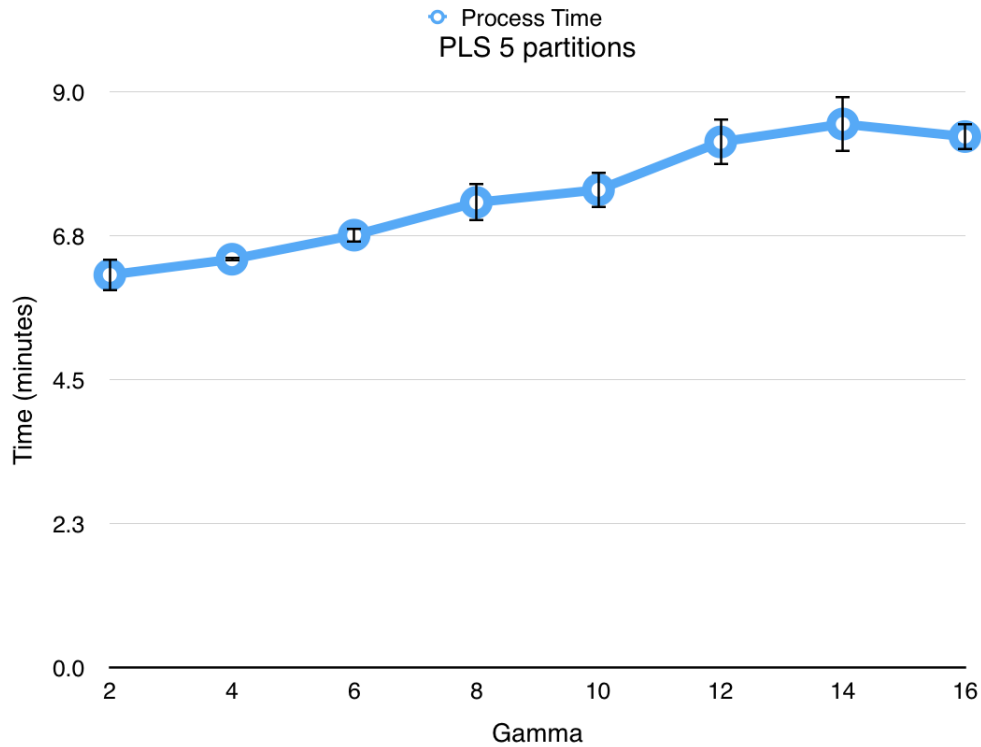


Figura 19: PLS-DA data set training

Por otro lado en esta imagen se puede ver que el tiempo de clasificación de una nueva imagen tiene un comportamiento bastante estacionario. Se ve que entre gamma 6 y 8 el costo temporal aumenta rápidamente y luego se mantiene.

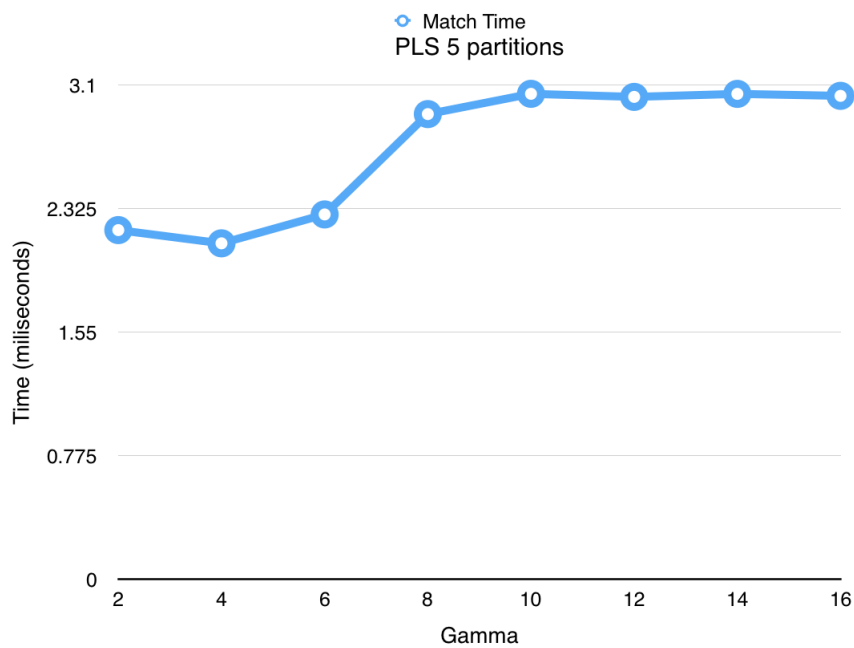
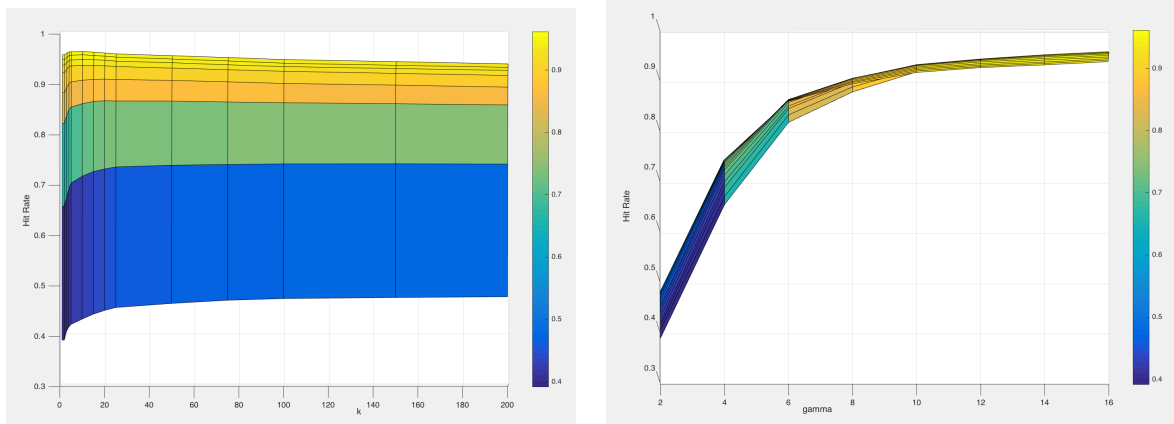


Figura 20: PLS-DA data set training

### Calidad del algoritmo

Para analizar la calidad de utilizar PLS-DA para reducir la dimensión de KNN tomamos el mismo rango de valores de  $\gamma$  y  $k$  y analizamos su hit rate obteniendo los siguientes gráficos.



Cuadro 10: Hit Rate para valores de  $\gamma$  y  $k$

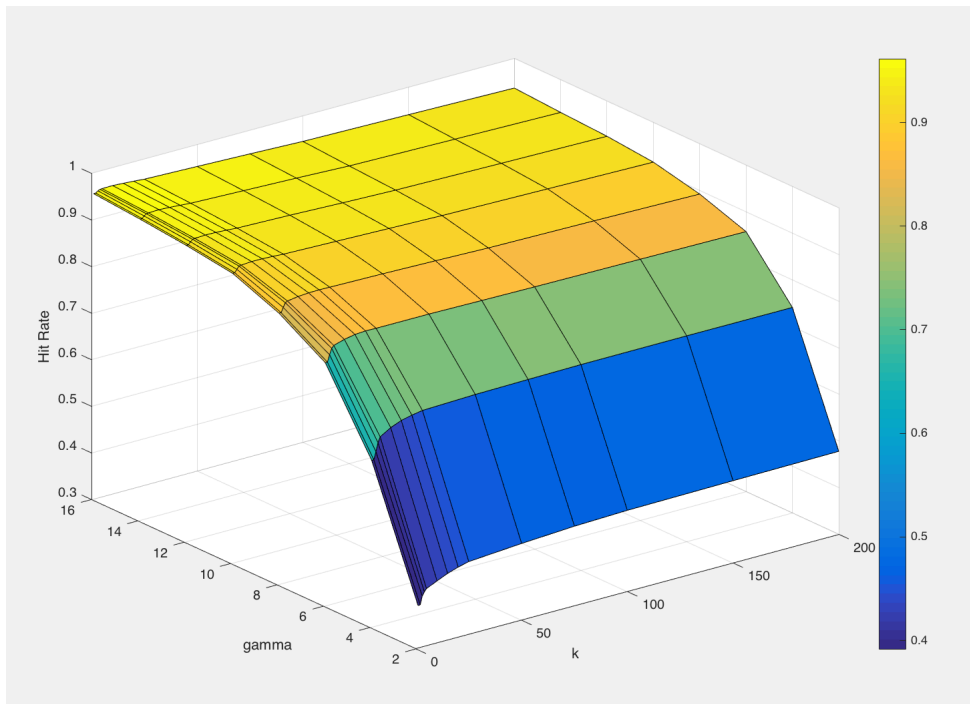
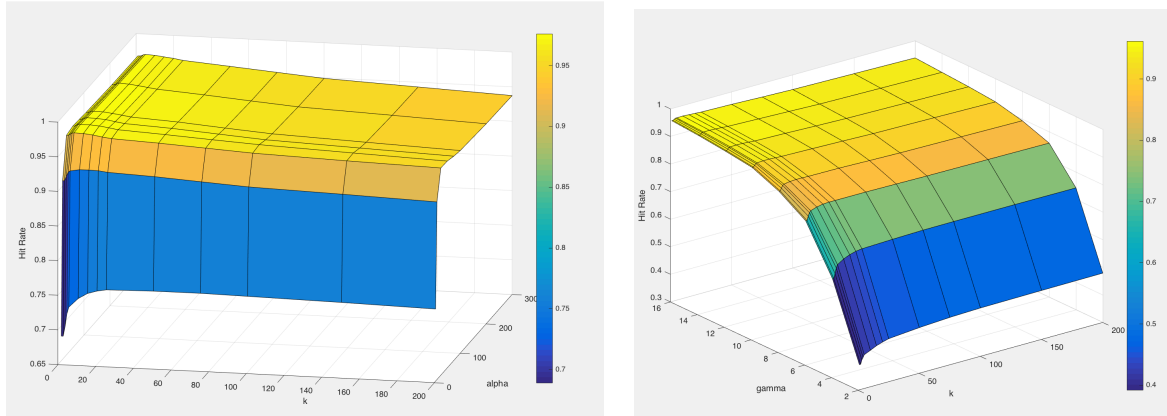


Figura 21: PLS-DA data set training

Como se puede observar en el primer gráfico, el comportamiento según  $k$  es bastante similar: primero aumenta y después comienza a descender, obteniendo un máximo en  $k = 10$ . Respecto al valor de  $\gamma$ , a diferencia del caso anterior la curva es estrictamente creciente. Aumentar  $\gamma$  siempre aporta información útil al algoritmo para ayudarlo a clasificar, por eso es más difícil establecer un criterio de parada. De todos modos, dado que cada vez crece más lento, vemos que con el  $k$  elegido, con  $\gamma = 14$  obtenemos un hit rate de 0.96. Se podría aumentar más el valor de  $\gamma$ , pero los incrementos son cada vez menores y el compromiso de tiempo es cada vez mayor.

#### 3.4.4. Cross Validation

Por último realizamos una comparación de estos datos con nuestra partición de  $K = 10$ .



Cuadro 11: Hit Rate para valores de  $\gamma$  y  $k$

La primera imagen mide el hit rate para PCA con  $\alpha$  y  $k$ , mientras que la segunda lo mide para PLS con  $\gamma$  y  $k$ . Se ve fácilmente que la forma de los planos es muy similar a la que los algoritmos tenían para  $K = 5$ . En particular el punto máximo para PCA sigue estando en  $\alpha = 50$  y  $k = 4$  mientras que PLS mantiene su comportamiento creciente en  $k = 10$ . Por otro lado, al igual que pasaba con KNN puro, utilizar 10 particiones nos da un ligero mejor hit rate ya que su base de entrenamiento es mayor.

## 4. Conclusiones

La conclusión mas notable es la *gran* reducción de tiempos de ejecución, con alta efectividad, que proveen los algoritmos *PCA* y *PLS-DA*. Las gráficas mostradas son contundentes en este sentido. Esto es de vital importancia, sobre todo por el hecho de que se podría aplicar, casi sin esfuerzo, a dígitos manuscritos de mayor resolución y calidad (lo que genera un aumento en la cantidad de píxeles) y solventar el incremento fuerte que sufriría el costo temporal de aplicar *kNN* plano.

A su vez, pudimos comprobar que, por lo menos para el problema planteado, existen “representantes” de la información que distinguen los elementos entre sí. Si bien estos representantes se obtenían en un nuevo espacio de los datos, los mismos estaban intrínsecamente relacionados con la entrada original. Esto se puede ver, por ejemplo, en los *autodígitos* obtenidos (3.2). Inclusive hubo casos (3.3) dónde introducir una mayor cantidad de variables puede disminuir (aunque sea levemente) la precisión de la clasificación. Esto se debe a que a mayor información a ser tenida en cuenta, mayores son las chances de que esa *data* sea poseída por 2 o más elementos y por ende generar más similitud entre los elementos transformados, que se traduce en confusiones a la hora de clasificar.

Es claro que un elemento fundamental que habilita la buena aplicación del algoritmo es la base de datos sobre la cuál se trabaja. Por ende, la importancia a la hora de plantear esta metodología de clasificación yace en 2 frentes: la correcta y eficiente algoritmia, y un conjunto de *entrenamiento* abarcativo y representativo. No puede flaquear ninguno de los 2 elementos a la hora de construir una resolución de este estilo.

De todas formas, tal vez lo más interesante sea el concepto general detrás de los métodos: poder generar un nuevo universo de información, relacionada con la original, en el cuál se pueda facilitar la ponderación de la información y reducir la dimensión de la entrada para agilizar su clasificación. Esto, probablemente, trascienda el contexto de este trabajo y sea aplicable a problemas de características similares. Encontramos este *approach* ciertamente contra-intuitivo lo que magnifica, por lo menos a nuestro entender, su efectividad.



## 5. Extras

Aquí pondremos algunos resultados/planteos interesantes.

### 5.1. Veredicto de *Kaggle*

El primero fue que corrimos nuestros algoritmos para los valores “óptimos” que obtuvimos de nuestro análisis, utilizando la base de train completa para entrenar, y luego etiquetamos las imágenes provistas en la competencia de Kaggle como test.

Método	#Autovalores	#Vecinos	Hit-Rate
PCA	50	4	0.97214
PLS-DA	16	10	0.95971
PCA	75	4	0.97300
PLS-DA	18	10	0.96186
PCA	85	4	0.97243
PLS-DA	50	10	0.97143

Cuadro 12: Corridas en Kaggle

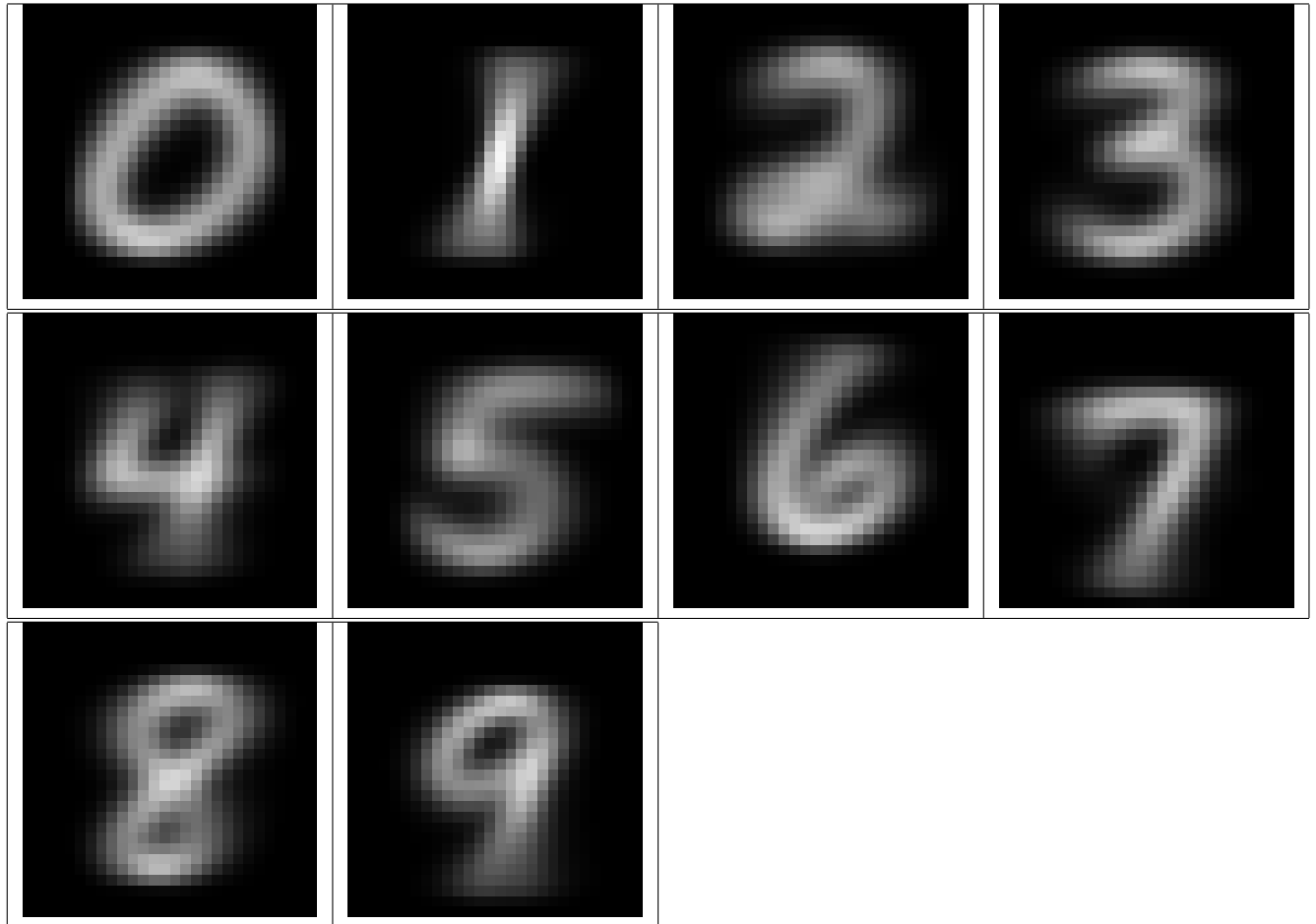
Habíamos obtenido como óptimos en relación *Tiempo* – *HitRate* los primeros dos valores de la tabla. Pero dado que la competencia la gana el mayor Hit Rate, decidimos realizar una corrida con los parámetros de las siguientes entradas de la tabla anterior.

El mayor valor obtenido fue con el método PCA con  $\alpha = 75$  y  $k = 4$ .

PLS-DA corrido con gammas más grandes, aumentan el Hit Rate, pero el tiempo de cómputo es abismalmente mayor.

### 5.2. Alternativa de clasificación: *Promedio* de imágenes

Una idea que surgió en pleno trabajo fue: “¿Habría alguna manera que sea simultáneamente simple y efectiva de clasificar?” Con esta pregunta en mente, calculamos el promedio de cada dígito de la base de *train* completa y observamos los resultados:



Esto nos dice algunas cosas:

Primero que todo, que la base de entrenamiento sobre la cuál trabajamos es abarcativa. El hecho de, a partir de haber calculado los promedios sobre la misma, hayamos obtenido imágenes que tienen gran similitud con potenciales números manuscritos lo respalda.

Otra observación/conjetura que nos surgió es que, probablemente, este conjunto de imágenes podría ser utilizado para realizar un algoritmo clasificador más simple que conste de calcular la *distancia* una imagen a clasificar contra este conjunto particular, y *taggear* en base a la que se encuentre mas cercana. Este procedimiento, en comparación a los métodos postulados en el trabajo, es de una simpleza notablemente mayor. De ser efectivo este algoritmo, su simpleza podría ser útil en contextos dónde no se tenga fácil acceso a algoritmos como *PCA* o *PLS-DA*, como competencias, lenguajes de programación poco maduros o con poca orientación científica, etc.

De todas formas, la pregunta planteada no fue del todo respondida puesto que no tenemos datos que la fundamenten. Queda pendiente obtener métricas de calidad en base a este planteo y llegar a una conclusión en base a los resultados.

De ser efectivo, esto podría disparar otras potenciales preguntas como: “¿Lo será también en otros problemas de clasificación con datos *menos* acotados como clasificación de caras?”, “¿Se podrá *simular* la caligrafía de una persona a partir de los *promedios* de sus símbolos lexicográficos manuscritos?”, “¿De qué depende su efectividad?”. Queda para el interesado experimentar con estos disparadores.

## 6. Referencias

[1] Análisis Numérico, página 491 - Burden & Faires