# Reconocimiento de Escritura Manuscrita

***(Online Handwriting Recognition)***

*Tesina de Grado*

*Septiembre 2011*

PABLO SPECIALE



*Lic. en Cs. de la Computación*
*Facultad de Ciencias Exactas, Ingeniería y Agrimensura*
*Universidad Nacional de Rosario*

**Director**: Dr. Juan Carlos Gomez[1]

**Co-director**: Dr. Pablo Granitto[2]

[1]Director del grupo *Procesamiento de Señales Multimedia* del CIFASIS
[2]Director del grupo *Aprendizaje Automatizado y Aplicaciones* del CIFASIS

# Resumen

Este trabajo tiene como objetivo reconocer escritura manuscrita obtenida digitalmente como secuencias de puntos de una manera robusta; ésto es, reconocer trazos indistintamente de que sean dígitos, letras o símbolos matemáticos. Se probaron diferentes métodos basados en la misma idea: tratar las secuencias de puntos como curvas continuas, lo cual es posible aproximando los trazos mediante bases de polinomios ortogonales. Se probará que dichas aproximaciones caracterizan muy bien a los trazos, permitiendo alcanzar una alta precisión en el reconocimiento, y eficiencia computacional. Se obtuvieron buenos resultados en dos bases de datos diferentes, una de dígitos y otra de letras.

# Agradecimientos

Una canción que me gusta mucho últimamente es: "**My way**", de *Frank Sinatra*. Quizás por haber descubierto que es parte de mi filosofía, quizás por la forma en que la escuché por primera vez, o quizás simplemente porque me gusta. Lejos de saber el porqué de mi gusto, les dejo un fragmento:

*« Tal vez lloré, tal vez reí,*
*tal vez gané, tal vez perdí,*
*ahora sé que fui feliz,*
*que si lloré también amé,*
*todo ello fue, puedo decir,*
*a mi manera.*

•

*Quizás también dudé,*
*cuando mejor me divertía;*
*quizás yo desprecié,*
*aquello que no comprendía,*
*hoy sé qué firme fui,*
*y que afronté ser como era,*
*y así logré seguir,*
*a mi manera.*

•

*Porque ya sabrás,*
*que el hombre al fin,*
*conocerás por su vivir,*
*no hay porque hablar, ni que decir,*
*ni recordar, ni hay que fingir,*
*puedo llegar, hasta el final,*
*a mi manera! »*

Vivir a "mi manera" ha sido posible gracias a encontrarme por el camino con **gente maravillosa**, que siempre me ha alentado para que me sienta libre. Agradecimientos:

A **mis directores**. A Juan Carlos y Pablo, por permitirme trabajar en el tema que elegí y guiarme para finalizarlo.

A **mis profesores de la universidad**. A Mariano por transmitir su amor a la ciencia. A Graciela por sus acertados consejos. A Guido, Maxi, Silvia y Valeria por compartir su conocimiento.

A **mis amigos de la facultad**. A Pablito, por llevar alegría adonde vaya. A Mauricio y Martín, por demostrarme que hay otros caminos. A Rodrigo, por el humor de todos los días y de sus constantes palabras de aliento. A Franco, por su ayuda con inglés. A Daniel, por el esfuerzo compartido en una difícil etapa. A Guille, por los partidos de tenis.

A **mis amigos de la secundaria**. A Pablo, por compartir desde fútbol, la primaria, la secundaria, hasta parte de la facultad. A Leandro y Damián, por ser (sin saberlos ellos) mis ídolos, por cómo encaran la vida.

A **los pibes del barrio**. A Martín y Esteban, por entenderme desde el principio, grandes amigos de la vida.

A la gente que me ha dado ese **empujoncito** que necesitaba. A Mario Munich, por permitirme trabajar en Evolution, en donde encontré el tema que me gusta. A Ana por siempre alentarme a que termine. A Cintia, por darme el cielo y el infierno, lástima que el orden no sea conmutativo, pero gracias por haberme hecho sentir vivo.

A **mi familia**. A Tomy (mi sobrinito), por enseñarme que la felicidad pasa por otro lado, y que las cosas simples pero del corazón son las más importantes. A Romina (mi hermana), eternamente agradecido por soportarme y alentarme todo este tiempo. A Cristina (mi mamá) y Roberto (mi papá), por darme la vida y ser los principales artífices de ser como soy, les debo todo, los quiero mucho.

A todos, a los que mencioné y a los que olvidé mencionar... ¡Muchas gracias por permitirme vivir... **A Mi Manera**!

# Índice

# 1. Introducción

## 1.1. Motivación

Actualmente es posible escribir en dispositivos electrónicos, sobre todo con la llegada de las Tablet PC, pizarras eléctricas, celulares touch-screen y las pantallas táctiles. También cabe destacar algunos e-Readers que permiten la escritura, convirtiéndose en *papel electrónico*. La escritura se realiza generalmente con un *stylus* (lápiz), abriendo la puerta a nuevas interacciones más allá del teclado y mouse. En la figura 1, puede apreciarse una variedad de dispositivos que permiten la escritura con lápiz.


**Figura 1:** Tabletas

A pesar de que exista una variedad importante de tales dispositivos, no hay todavía una aplicación sobresaliente de reconocimiento para ellos. El usuario ve al *stylus* como un mouse sofisticado, sin que se logre una mejora significativa en la productividad al no explotar la potencialidad del mismo. Un buen sistema de reconocimiento de escritura (*handwriting recognition*[3]) podría permitir al usuario una mejor experiencia con respecto al papel y lápiz tradicional, permitiéndole obtener resultados inmediatos a partir de su escritura, como ser resultados matemáticos.

## 1.2. Organización del trabajo

En la sección 2, se introducen los conceptos generales de reconocimiento de escritura; en la sección 3, se introducen los fundamentos teóricos en los que se basa el trabajo; en la sección 4, se explica la extracción de características; en la sección 5, se comentan los métodos de clasificación utilizados; en la sección 6, se comparan todos los métodos; en la sección 7; se provee una visión futura de las posibles aplicaciones de los métodos explicados; y por último, en la sección 8, se presentan algunas conclusiones.

[3]Muchos términos utilizados aquí se dejan intencionalmente en inglés, facilitando la búsqueda de contenido para tales términos.

# 2. Conceptos Generales

En esta sección se describirán las partes esenciales de sistema de reconocimiento de escritura manuscrita, permitiendo así aclarar en dónde se centrará este trabajo. También se comentarán brevemente las etapas que intervienen en el proceso de reconocimiento de escritura.

## 2.1. Reconocimiento Online vs Offline

A diferencia del enfoque *Offline* de handwriting recognition, el cual pretende reconocer caracteres a partir de una imagen, el enfoque *Online* intenta el reconocimiento a partir de secuencias de puntos, aquí llamados trazos (*strokes*). Ambos enfoques están esquemáticamente representados en la figura 2.

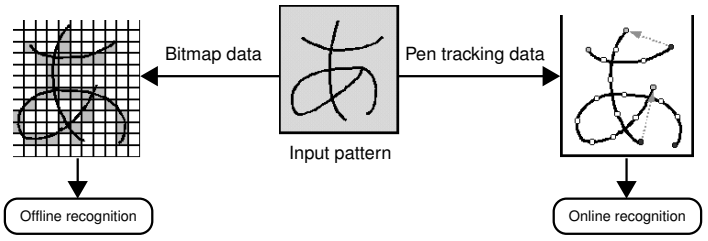



**Figura 2:** Offline vs Online.

Como puede verse, en el enfoque, se conoce el orden en que cada uno de los puntos fue escrito. Así, es posible diferenciar distintos estilos de escrituras, a pesar de que el resultado final puede que sea el mismo. La utilización de un stylus en un dispositivo electrónico permite usar el enfoque Online.

## 2.2. Segmentación

En el presente trabajo, se asume que la escritura está en imprenta y que los símbolos están bien distanciados entre sí. En la figura 3, se muestran distintos niveles de dificultad en la segmentación, variando desde el caso más simple (el ya segmentado, caso 1) hasta el caso más complejo (mezcla de escritura cursiva e imprenta, caso 5). Se considera el caso 2 para la construcción de la base de datos. Los niveles superiores de escritura requieren de un trabajo adicional de segmentación que no entra en el alcance de este trabajo.

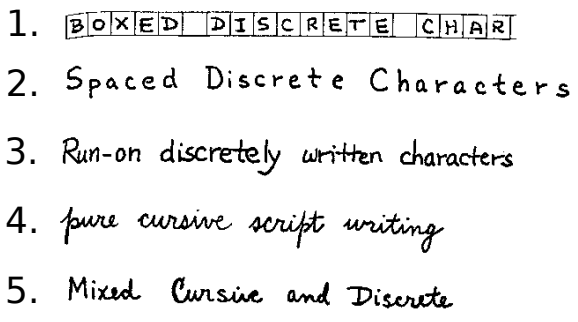



**Figura 3:** Niveles de dificultad en segmentación [Connell 00].

## 2.3. Etapas del Reconocimiento de Escritura

En el análisis de escritura manuscrita se pueden reconocerse las siguientes posibles etapas:

### 2.3.1. Colección de trazos

Un símbolo[4] puede ser escrito con un sólo trazo (*Single-Stroke*) o con muchos (*Multi-Stroke*), ver figura 4. El problema de determinar qué trazos pertenecen a qué caracteres es llamado *Stroke grouping*. Observar que esta etapa depende de la segmentación asumida, ver sección 2.2.

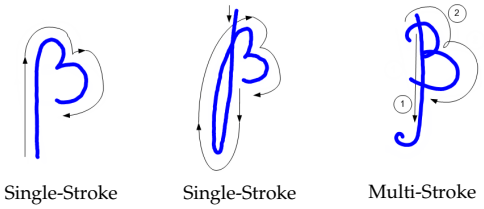



**Figura 4:** Los dos primeros trazos son Single-Stroke, y el último es Multi-Stroke.

### 2.3.2. Preproceso

En el preprocesado pueden aplicarse diferentes tipos de filtros. El *suavizado* elimina el ruido frecuentemente presente al principio y final de cada trazo. La normalización por *resampling* ayuda a unificar distancias entre los puntos, y la normalización por *resizing* asegura que el tamaño del caracter escrito no afecte al resultado del reconocimiento. En la figura 5 puede observarse una posible secuencia de aplicación de filtros.

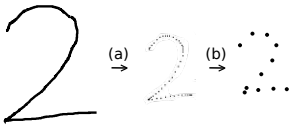



**Figura 5:** `(a)` Suavizado y Resizing, `(b)` Resampling

### 2.3.3. Extracción de características (*Feature extraction*)

En este paso se extraen características (*features*) de los trazos. Hay varios tipos de features que se han usado tradicionalmente: los relativos a la apariencia que incluye, por ejemplo, proporción entre altura y ancho; los relativos a propiedades geométricas, como ser la cantidad de loops e intersecciones, centro de gravedad, puntos dominantes; los relativos al estilo de escritura, como la dirección y cantidad de strokes. Se posterga para la sección 4 la explicación de los features utilizados en este trabajo.

[4]Se usa el término *símbolo* en vez de *caracter* pues puede ser un dígito, una letra o un símbolo matemático.

### 2.3.4. Entrenamiento (*Training*)

Se requiere tener una base de datos de modelos con la cual contrastar el símbolo actual de entrada. Se realiza por una única vez en los algoritmos que permiten entrenamiento.

### 2.3.5. Reconocimiento o Estimación (*Matching*)

Aquí se intenta estimar, para la entrada actual, cuál modelo es el más "similar", sobre la base de datos disponible que sirvió de entrenamiento. Es decir, se clasifica a la nueva entrada determinando a qué clase pertenece. Pueden considerarse múltiples estimaciones, por ejemplo, dar las mejores tres estimaciones; pero en este trabajo solo se considera la primera (*first match*).

## 2.4. Dependiente o independiente del usuario

Un sistema de *handwriting recognition* puede ser dividido en: dependiente del usuario (*writer-dependent*) o independiente del usuario (*writer-independent*). Un sistema *writer-independent* es entrenado para reconocer una gran variedad de estilos de escritura, mientras que un sistema *writer-dependent* es entrenado para reconocer a un único individuo. Un sistema *writer-dependent* trabaja con datos con poca variabilidad, y entonces puede alcanzar mejores resultados, el único problema es que exige al usuario tomarse el tiempo para entrenar el sistema.

En este trabajo se pueden utilizar ambos enfoques, dependiendo de la base de datos utilizada.

# 3. Fundamentos

En esta sección se introducirán algunos conceptos que utilizaremos a lo largo del trabajo.

## 3.1. Preliminares

Sea una familia de funciones $\{B_i\}$ en el espacio de funciones continuas en $[a, b]$, se definen

**Producto interno:**

$$\langle B_i, B_j \rangle \doteq \int_a^b B_i(t)\, B_j(t)\, w(t) dt, \tag{1}$$

donde $w(t)$ es una función peso.

**Norma inducida:**

$$\|B_i\| \doteq \sqrt{\langle B_i, B_i \rangle} \tag{2}$$

**Kronecker delta:**

$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases} \tag{3}$$

**Ortogonalidad:**

Se dice que $B_i$ y $B_j$ son ortogonales si se cumple que $\langle B_i, B_j \rangle = 0, \qquad \forall i \neq j$.
Si además, $\langle B_i, B_i \rangle = 1$ se dice que $B_i$ y $B_j$ son ortonormales.

**Proceso de ortogonalización de Gram-Schmidt:**

Sea $\{v_1, \ldots, v_n\}$ una base de un espacio $U$ con producto interno, se define recursivamente

$$u_i = \left( v_i - \sum_{j=1}^{i-1} \langle v_i, u_j \rangle u_j \right), \qquad i = 1, 2, \ldots, n \tag{4}$$

Entonces, $\{u_1, \ldots, u_n\}$ es una base ortogonal y $\left\{ \frac{u_1}{\|u_1\|}, \ldots, \frac{u_n}{\|u_n\|} \right\}$ es base ortonormal de $U$.

## 3.2. Polinomios ortogonales

Polinomios ortogonales son clases de polinomios $\{B_i(t)\}$ definidos sobre el dominio $[a, b]$ que obedecen la relación de ortogonalidad

$$\int_a^b B_i(t)\, B_j(t)\, w(t) dt = 0, \qquad \text{cuando } i \neq j \tag{5}$$

donde $w(t)$ es una función peso. Si además cumplen con

$$\int_a^b B_i(t)\, B_j(t)\, w(t) dt = \delta_{ij} \tag{6}$$

los polinomios son ***ortonormales***. Es decir, están normalizados: $\|B_i\| = 1$

### 3.2.1. Polinomios de Legendre

Si tomamos como función peso $w(t) = 1$ en la definición (1) de producto interno, es decir,

$$\langle L_i, L_j \rangle = \int_{-1}^{1} L_i(t)\, L_j(t)\, w(t)\, dt = \int_{-1}^{1} L_i(t)\, L_j(t)\, dt$$

se pueden generar los polinomios de Legendre $\{L_i\}$ con el proceso de ortogonalización de Gram-Schmidt (4) en el intervalo $[-1, 1]$. Normalizando de tal manera que $L_n(1) = 1$ da los polinomios de Legendre esperados. Algunos de ellos son:

$$\begin{aligned} L_0(t) &= 1 \\ L_1(t) &= t \\ L_2(t) &= \frac{3\,t^2 - 1}{2} \\ L_3(t) &= \frac{5\,t^3 - 3\,t}{2} \\ L_4(t) &= \frac{35\,t^4 - 30\,t^2 + 3}{8} \\ &\vdots \end{aligned}$$

Para nuestro propósito es conveniente utilizar los polinomios análogos definidos en $[0, 1]$, los cuales son llamados ***shifted*** *Legendre polynomials*,

$$L_i(t) = \sum_{j=0}^{i} C_{ij}\, t^j \tag{7}$$

el grado de $L_i$ es exactamente $i$. Se conoce una fórmula cerrada para el cálculo de los coeficientes $C_{ij}$ ortonormales. A continuación se muestra dicha fórmula, ver [Abramowitz 70],

$$C_{ij} = (2\,i + 1)^{\frac{1}{2}} \binom{i}{j} (-1)^j \binom{j + i}{j} \tag{8}$$

Pueden verse los primeros diez polinomios de Legendre gráficamente en la figura 6.

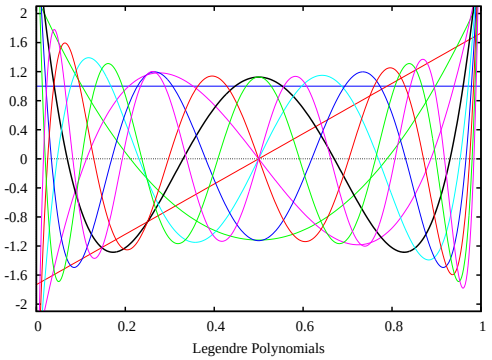



**Figura 6:** Primeros diez polinomios de Legendre (shifted) en $[0, 1]$

### 3.2.2. Polinomios de Chebyshev

Otros polinomios ortogonales que pueden usarse son los polinomios de Chebyshev. A diferencia de los de Legendre, la función peso es $w(t) = (1 - x^2)^{-\frac{1}{2}}$. Se pueden generar estos polinomios con el proceso de ortogonalización de Gram-Schmidt. Se muestran en la figura 7 algunos de ellos.

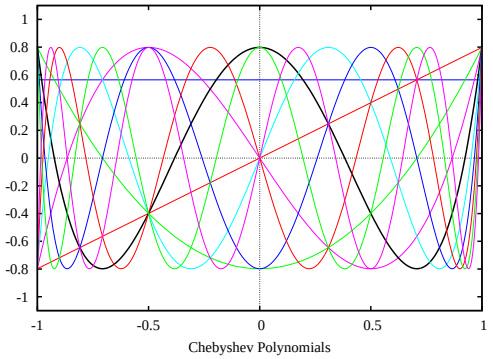


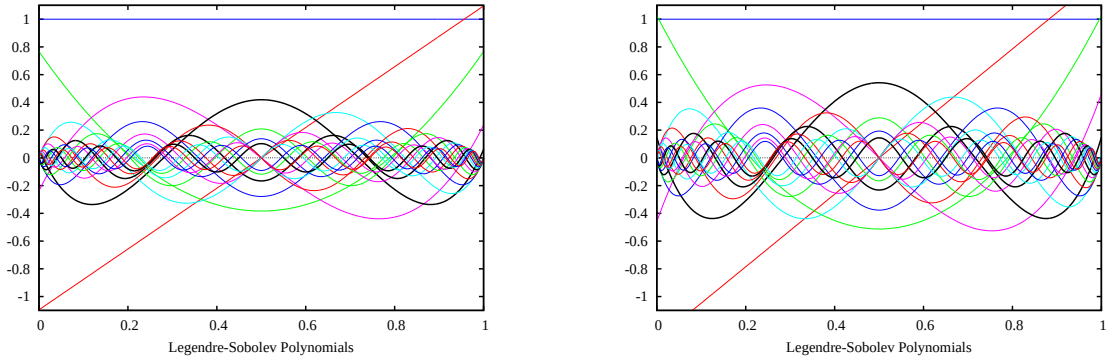

**Figura 7:** Primeros diez polinomios de Chebyshev, en el intervalo $[-1, 1]$

### 3.2.3. Polinomios de Legendre-Sobolev

Se puede utilizar el producto interno de Sobolev con peso $w(t) = 1$, es decir,

$$\langle f, g \rangle_{LS} = \int_a^b f(t)\, g(t)\, dt + \mu \int_a^b f'(t)\, g'(t)\, dt$$

dando lugar a los polinomios de Legendre-Sobolev, ver [Watt 09]. En la figura 8 pueden verse dos instancias con $\mu = \frac{1}{8}$ y $\mu = \frac{1}{16}$,




**(a)** $\mu = \frac{1}{8}$ **(b)** $\mu = \frac{1}{16}$

**Figura 8:** Primeros veinte polinomios de Legendre-Sobolev, en el intervalo $[0, 1]$

## 3.3. Series de funciones ortogonales

Si $\{B_i\}$ es una base de polinomios ortonormales respecto al producto interno (1), una función continua $f$ en el dominio $[a, b]$ puede ser escrita como

$$f(t) = \sum_{i=0}^{\infty} \alpha_i \, B_i(t)$$

Al aplicar el producto interno con $B_i$ en ambos lados de la ecuación anterior se obtiene:

$$\langle f, B_i \rangle = \left\langle \sum_{i=0}^{\infty} \alpha_i \, B_i(t), B_i(t) \right\rangle = \sum_{i=0}^{\infty} \alpha_i \langle B_i, B_i \rangle = \sum_{i=0}^{\infty} \alpha_i \delta_{ij} = \alpha_i$$

Reescribiendo,

$$\alpha_i = \langle f, B_i \rangle \tag{9}$$

Entonces, se puede obtener una aproximación a $f(t)$ al truncar la serie en un cierto orden $d$,

$$f(t) \approx \sum_{i=0}^{d} \alpha_i \, B_i(t) \tag{10}$$

# 4. Feature extraction

Las técnicas usuales para *handwriting recognition* tratan de encontrar *features* particulares sobre un conjunto de símbolos, citemos por ejemplo los dígitos. Pero al cambiar dicho conjunto, estos features dejan de ser efectivos, no discriminan correctamente. Se vuelve impráctico desarrollar heurísticas para reconocer features específicos para cada símbolo. Por lo que es deseable buscar una representación que permita ser aplicada sin importar qué tipo de símbolo se trate; ya sea un dígito, una letra o un símbolo matemático.

Unos de los mayores problemas con los métodos de reconocimiento tradicionales es que los trazos son tratados como secuencias de puntos (en *discreto*), en vez de verlos como lo que realmente son, curvas (en *continuo*).

## 4.1. Trazos discretos como curvas continuas

En vez de describir a los trazos como una secuencia de puntos, éstos pueden ser representados por aproximaciones de curvas, figura 9. Se mostrará que se necesitan menos de veinte (20) coeficientes de una serie para representar un trazo.

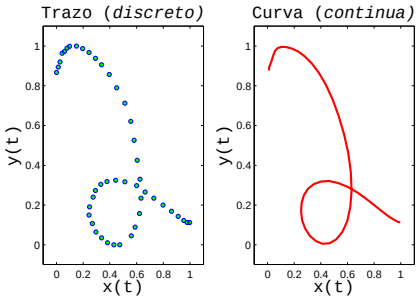



**Figura 9:** Trazos como curvas paramétricas: $r(t) = \{x(t), y(t)\}$

Observar que se necesitan aproximar dos curvas: $x(t)$, $y(t)$ por símbolo, figura 10.

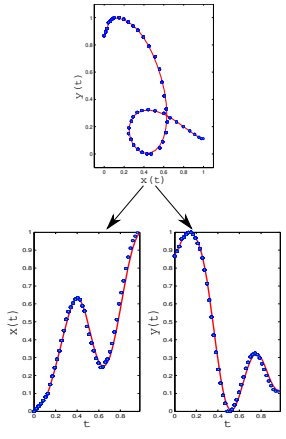


**Figura 10:** En rojo las aproximaciones de $x(t)$, $y(t)$

Trabajos anteriores [Char 07] han demostrado cómo las coordenadas $x(t)$, $y(t)$ de símbolos escritos a mano pueden ser representados como expansión en series de polinomios ortogonales

de *Chebyshev*; y que los coeficientes de las series truncadas además de ser las aproximaciones buscadas, pueden ser usados para clasificación y reconocimiento.

De manera similar, en este trabajo se utiliza un número finito de ***momentos matemáticos*** para la reconstrucción de funciones (las curvas) como series truncadas de polinomios ortogonales de ***Legendre***, siguiendo las ideas presentadas en los artículos [Talenti 87] y [Golubitsky 08]. Este problema es conocido como ***Hausdorff Moment Problem*** [Hausdorff 21]. Una alternativa a este método para la obtención de las aproximaciones a las curvas (es decir, los coeficientes de las series truncadas) es utilizar *aproximación por* ***mínimos cuadrados*** a través de la ***pseudo-inversa*** *de Moore-Penrose*, permitiendo este enfoque utilizar polinomios ortogonales cualesquiera; en particular, polinomios de ***Legendre***, ***Legendre-Sobolev*** y ***Chebyshev***.

## 4.2. Representación con series

Se desea entonces aproximar las funciones $x(t), y(t)$ de los trazos. Como se indicó en la sección 3.3, ésto puede hacerse como expansión de series de polinomios ortogonales; según la ecuación (9),

$$\begin{cases} x(t) & \approx & \sum_{i=0}^{d} \alpha_i \, B_i(t) \\ y(t) & \approx & \sum_{i=0}^{d} \beta_i \, B_i(t) \end{cases} \tag{11}$$

La clasificación puede ser obtenida, por ejemplo, al medir la distancia Euclidiana de los vectores $(\alpha_0, \ldots, \alpha_d, \beta_0, \ldots, \beta_d)$ entre los distintos trazos.

Consideraremos **features** a estos vectores que tienen dimensión $2\,(d+1)$. En lo siguiente, se mostrarán dos formas de calcularlos y, en secciones posteriores, resultados que avalen que los features son lo suficientemente representativos como para clasificar de manera precisa a los trazos.

## 4.3. Momentos

Una posible forma de calcular los $\{\alpha_i\}$ y $\{\beta_i\}$ es mediante momentos, como se explicará a continuación.

### 4.3.1. Definición

Los momentos matemáticos de una función $f$ definida en $[0, 1]$ son

$$\mu_k \doteq \int_0^1 f(\lambda) \, \lambda^k \, d\lambda \tag{12}$$

### 4.3.2. Hausdorff Moment Problem

Este problema consiste en recuperar a partir de una secuencia finita de momentos $\{\mu_k\}_{k=0,1,2,\ldots}$ una función $f$ en el dominio $[0, 1]$.

Con el fin de facilitar el desarrollo, recordaremos algunas definiciones y ecuaciones previas. Se supondrá que el dominio es $[0, 1]$, y que la función peso es $w(t) = 1$ para poder trabajar con

los polinomios de *Legendre* (shifted), sección 3.2.1. Por lo que el producto interno en (1) queda

$$\langle f, g \rangle = \int_0^1 f(t)\, g(t)\, dt \tag{13}$$

Escribamos nuevamente la ecuación (7),

$$L_i(t) = \sum_{j=0}^{i} C_{ij}\, t^j$$

y también la ecuación (10), pero con el reemplazo de $B_i$ por $L_i$,

$$f(t) \approx \sum_{i=0}^{d} \alpha_i\, L_i(t)$$

Ahora, a partir de

$$\begin{aligned}
& \alpha_i \\
= \quad & \langle \text{ por (9) } \rangle \\
& \langle f, L_i \rangle \\
= \quad & \langle \text{ expansión del producto interno (13) } \rangle \\
& \int_0^1 f(t)\, L_i(t)\, dt \\
= \quad & \langle \text{ definición de } L_i(t) \text{, ecuación (7) } \rangle \\
& \int_0^1 f(t) \left( \sum_{j=0}^{i} C_{ij}\, t^j \right) dt \\
= \quad & \langle \text{ reordenación de términos, sumatoria afuera } \rangle \\
& \sum_{j=0}^{i} C_{ij} \underbrace{\left( \int_0^1 f(t)\, t^j\, dt \right)}_{\mu_j} \\
= \quad & \langle \text{ definición de momento (12) } \rangle \\
& \sum_{j=0}^{i} C_{ij}\, \mu_j
\end{aligned}$$

Se puede escribir en forma matricial

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix}_{(d+1)} = \begin{bmatrix} C_{00} & & & \\ C_{10} & C_{11} & & \\ \vdots & \vdots & \ddots & \\ C_{d0} & C_{d1} & \dots & C_{dd} \end{bmatrix}_{(d+1)\times(d+1)} \cdot \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_d \end{bmatrix}_{(d+1)}$$

Resumiendo,

$$\alpha = \mathbf{C}\, \mu \tag{14}$$

donde $\mathbf{C}$ es la matriz de coeficientes de los polinomios de Legendre, y $\mu$ un vector columna con los momentos matemáticos de la función a reconstruir. De esta manera, se tiene que el cálculo de $\alpha$, los coeficientes de la expansión finita por series de la función $f$, se realiza a partir de una multiplicación matricial en la que intervienen los momentos.

Observar que en la ecuación anterior (14), la matriz $\mathbf{C}$ puede ser **precalculada** con la fórmula cerrada (8), o mediante el proceso de Gram-Schmidt (4). Cualquiera sea la forma en que se calculen los coeficientes de los polinomios de Legendre, es importante notar que solo es necesario realizarlo por única vez, por lo que es necesario focalizarse en el cálculo de los momentos.

### 4.3.3. Momentos para funciones definidas en otros dominios

Generalicemos la definición (12), sea $f$ una función definida en $[0, \ell]$ los momentos de $f$ son

$$\mu_k(f, \ell) = \int_0^\ell f(\lambda)\, \lambda^k \, d\lambda \tag{15}$$

Al estar los polinomios de Legendre (shifted) definidos en $[0, 1]$ es necesario calcular los momentos en ese dominio. Entonces, el problema aquí es cómo calcularlos a partir de una función $f$ definida en $[0, \ell]$. Esto se resuelve con un **cambio de variable** como se verá de inmediato.

Obsérvese que si se define,

$$\gamma \doteq \frac{1}{\ell}\lambda \quad \Longrightarrow \quad \lambda \in [0, \ell] \Leftrightarrow \gamma \in [0, 1] \tag{16}$$

también definamos,

$$\hat{f}(\gamma) \doteq f(\ell\,\gamma) \tag{17}$$

Ahora, partiendo de la definición (15),

$$\begin{aligned}
\mu_k(f, \ell) &= \int_0^\ell f(\lambda)\, \lambda^k \, d\lambda, && \textbf{cambio de variable: } \begin{cases} \lambda = \ell\,\gamma \\ d\lambda = \ell\, d\gamma \end{cases} \\
&= \int_0^1 f(\ell\,\gamma)\, (\ell\,\gamma)^k \, \ell \, d\gamma, && \text{reordenando} \\
&= \ell^{k+1} \int_0^1 f(\ell\,\gamma)\, \gamma^k \, d\gamma, && \text{por (17)} \\
&= \ell^{k+1} \underbrace{\int_0^1 \hat{f}(\gamma)\, \gamma^k \, d\gamma}_{\mu_k(\hat{f}, 1)}, && \text{por (15)} \\
&= \ell^{k+1}\, \mu_k(\hat{f}, 1)
\end{aligned}$$

obtenemos,

$$\mu_k(\hat{f}, 1) = \frac{\mu_k(f, \ell)}{\ell^{k+1}} \tag{18}$$

Por lo tanto, se utilizarán los momentos generalizados de la función $f$ original en $[0, \ell]$ para calcular los momentos de la función $\hat{f}$ en el dominio que se necesitan, es decir, $[0, 1]$.

### 4.3.4. Invariante a cambios de escala

Recordemos la ecuación (14),

$$\alpha = \mathbf{C}\,\mu$$

Se demostrará que la normalización de este vector, es decir $\frac{\alpha}{\|\alpha\|}$, es invariante a escala. Supongamos que tenemos una función $f$ y la multiplicamos por un factor positivo $s$, a la función resultante la llamaremos $\hat{f}$, es decir,

$$\hat{f}(t) = s\,f(t), \qquad s > 0$$

Veamos cómo esto afecta al cálculo de momentos,

$$\begin{aligned}\mu_k(\hat{f},\ell) &= \int_0^\ell \hat{f}(\lambda)\,\lambda^k\,d\lambda \\ &= \int_0^\ell s\,f(\lambda)\,\lambda^k\,d\lambda \\ &= s\,\mu_k(f,\ell)\end{aligned}$$

Utilizando (14) para el cálculo de $\hat{\alpha}$, correspondiente a la función $\hat{f}$

$$\hat{\alpha} = \mathbf{C}\,\hat{\mu} = \mathbf{C}\,(\mathbf{s}\,\mu) = s\,\mathbf{C}\,\mu = s\,\alpha$$

Recordando que $s > 0$, se obtiene

$$\frac{\hat{\alpha}}{\|\hat{\alpha}\|} = \frac{s\,\alpha}{\|s\,\alpha\|} = \frac{s\,\alpha}{|s|\,\|\alpha\|} = \frac{\alpha}{\|\alpha\|}$$

Resumiendo,

$$\frac{\hat{\alpha}}{\|\hat{\alpha}\|} = \frac{\alpha}{\|\alpha\|} \tag{19}$$

Se puede concluir entonces que al normalizar los vectores a norma 1, los **features son invariante a escala**.

### 4.3.5. Cálculo numérico de los momentos

Se asume que muestras de $f$ se irán recibiendo en tiempo real a medida que el usuario realiza el trazo. En principio, se podría esperar a que el trazo sea finalizado para comenzar el cálculo de los momentos, pero se mostrará que es posible ir calculándolos a medida que los datos van llegando. De esta manera se logra que la extracción de features sea extremadamente eficiente, pues una vez que el usuario levanta su lápiz (finalización del trazo), los momentos matemáticos ya han sido calculados. Pero antes, nos enfocaremos en el cálculo numérico.

Obsérvese que la integral siguiente debe aproximarse

$$\mu_k(f,\ell) = \int_0^\ell f(\lambda)\,\lambda^k\,d\lambda$$

Se ha intentado con las reglas del ***Trapecio*** (primer orden) y ***Simpson*** (segundo orden), pero no se ha alcanzado suficiente precisión para los valores de $\mu_k$ cuando $k$ crece. Métodos que requieran el conocimiento de $\ell$ tampoco pueden ser aplicados, por lo explicado en el primer párrafo.

Un método adecuado es una instancia especializada del método de integración de primer orden, debido a [Golubitsky 08]. Para computar $\int_0^\ell f(\lambda)\,\lambda^k\,d\lambda$ se usa la siguiente aproximación en cada intervalo $[i, i+1]$,

$$\begin{aligned}\int_i^{i+1} f(\lambda)\,\lambda^k\,d\lambda &\approx \underbrace{\frac{f(i+1)+f(i)}{2}}_{\text{valor medio}} * \underbrace{\int_i^{i+1}\lambda^k\,d\lambda}_{\text{integral exacta}} \\ &= \frac{f(i+1)+f(i)}{2} * \left.\frac{\lambda^{k+1}}{k+1}\right|_i^{i+1} \\ &= \frac{f(i+1)+f(i)}{2} * \frac{(i+1)^{k+1}-(i)^k}{k+1} \qquad (20)\end{aligned}$$

Esta fórmula representa la integral exacta en $\lambda^k$ en $[i, i+1]$ multiplicado por el valor medio de la función entre esos puntos. Sumando los intervalos,

$$\begin{aligned}\mu_k(f,\ell) &= \int_0^\ell f(\lambda)\,\lambda^k\,d\lambda \\ &= \sum_{i=0}^{\ell-1}\int_i^{i+1} f(\lambda)\,\lambda^k\,d\lambda, \qquad \text{aplicando (20) se obtiene} \\ &\approx \sum_{i=0}^{\ell-1}\frac{f(i+1)+f(i)}{2} * \frac{(i+1)^{k+1}-(i)^k}{k+1} \qquad (21) \\ &= \sum_{i=1}^{\ell-1}\frac{f(i-1)+f(i+1)}{2} * \frac{i^{k+1}}{k+1} \quad + \quad \frac{f(\ell-1)+f(\ell)}{2} * \frac{\ell^{k+1}}{k+1} \qquad (22)\end{aligned}$$

La última igualdad puede obtenerse expandiendo los términos de la ecuación (21), sucede una situación similar a las series telescópicas en las cuales se anulan los términos intermedios, solo quedando los extremos. La cantidad de términos a calcular se reduce a la mitad, hecho importante en la búsqueda de mayor eficiencia computacional.

En la ecuación (22), se puede ir actualizando la suma $\sum_{i=1}^{\ell-1}(\dots)$ para todos los *órdenes* $k$ de $0$ a $d$ tan pronto como el valor de $f(i+1)$ esté disponible, permitiendo así **calcular los momentos a medida que se ingresa el trazo**.

## 4.4. Pseudo-inversa de Moore-Penrose

Otra forma de calcular los $\alpha$ de la ecuación (10) es a través de la pseudo-inversa de Moore-Penrose. Esta solución no solo funciona con los polinomios de Legendre sino también para polinomios ortogonales $\{L_i\}$ cualesquiera. Definamos la siguiente función:

$$\begin{aligned}p(x) &= \sum_{i=0}^{d}\alpha_i\,L_i(x) \\ &= \alpha_0\,L_0(x) + \alpha_1\,L_1(x) + \cdots + \alpha_d\,L_d(x)\end{aligned}$$

Planteamos el siguiente sistema sobre-determinado de ecuaciones

$$\begin{cases} p(x_0) &= \alpha_0\, L_0(x_0) + \alpha_1\, L_1(x_0) + \cdots + \alpha_d\, L_d(x_0) \\ p(x_1) &= \alpha_0\, L_0(x_1) + \alpha_1\, L_1(x_1) + \cdots + \alpha_d\, L_d(x_1) \\ \vdots & \\ p(x_d) &= \alpha_0\, L_0(x_d) + \alpha_1\, L_1(x_d) + \cdots + \alpha_d\, L_d(x_d) \end{cases}$$

que puede escribirse matricialmente como

$$\underbrace{\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{bmatrix}}_{P} = \underbrace{\begin{bmatrix} L_0(x_0) & L_1(x_0) & \dots & L_d(x_0) \\ L_0(x_1) & L_1(x_1) & \dots & L_d(x_1) \\ \vdots & \vdots & & \vdots \\ L_0(x_d) & L_1(x_d) & \dots & L_d(x_d) \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix}}_{\alpha}$$

$$P = L\,\alpha$$

Es sabido que la solución del siguiente problema de minimización

$$\alpha = \arg\min_{\alpha} \; \|P - L\alpha\|^2,$$

es la estima de mínimos cuadrados dada por

$$\begin{aligned} \hat{\alpha} &= (L^T L)^{-1} L\, P \\ &= L^{\dagger}\, P \end{aligned}$$

donde $L^{\dagger}$ es la pseudo-inversa de Moore-Penrose.

## 4.5. Parametrización por longitud de arco

Parametrización por tiempo y por longitud de arco (*arc-length*) son las elecciones más populares en el campo de reconocimiento de escritura online, ver [Nalwa 97] y [Golubitsky 10]. Parametrización por longitud de arco es usualmente preferible, pues no es afectada por variaciones en la velocidad de escritura. Esta puede ser expresada como:

$$\textit{arc-length}(t) = \int_0^t \sqrt{(x'(\lambda))^2 + (y'(\lambda))^2}\, d\lambda$$

## 4.6. Necesidad del preproceso

Se ha demostrado que el método es invariante a escala 4.3.4, y comentado en 4.5 que la parametrización por longitud de arco no es afectada por las variaciones en la velocidad de escritura. Pero no se ha mencionado que sea invariante a traslación, pues no lo es. Sí se puede probar empíricamente que a pesar de que no sea invariante a traslación, pequeñas traslaciones no perturban demasiado el cálculo de los vectores, obteniendo igualmente resultados razonables que pueden ser usados en otros escenarios en donde la eficiencia computacional es un requerimiento determinante, como ser los dispositivos móviles. De aquí se deriva el análisis de si es posible evitar el preproceso, el cual se posterga hasta la sección 6.5.

# 5. Clasificación

Sea $\mathcal{D}$ un conjunto de entrenamiento de la forma $\{(x_i, c_i)\}_{i=1}^{n}$, donde $c_i \in C = \{k_1, k_2, \ldots, k_p\}$ indica la clase de la entrada $x_i$, siendo $p$ el número total de clases. Se desea determinar la clase $\hat{c}$ de una nueva entrada $\hat{x}$, donde $\hat{x} \notin \mathcal{D}$. Este proceso se llama clasificación.

En esta sección se introducirán los métodos de clasificación usados. Recordar que nuestros ***features*** son vectores de dimensión $2\,(d+1)$, dados por la ecuación (11) de la sección 4.2, es decir, tienen la siguiente forma:

$$(\alpha_0, \ldots, \alpha_d, \beta_0, \ldots, \beta_d)$$

## 5.1. Vecinos más cercanos: $k$-NN

Este método intenta encontrar los $k$ vecinos más "cercanos" de un vector feature de entrada con respecto a los vectores features de una base de datos. El significado de "cercano" es asociado a una métrica. Las distancias que se usaron fueron: distancia ***Euclidiana***, de ***Hamming***[5] y de ***Mahalanobis***.

Las distancias se especifican definiendo el operador $dist(x, y)$, donde $x = (x_0, x_1, \ldots, x_d)$ e $y = (y_0, y_1, \ldots, y_d)$ son vectores de dimensión $d + 1$. Veamos a continuación las distancias utilizas en este trabajo.

### 5.1.1. Distancia Euclidiana

$$\begin{aligned} dist(x,y)^2 &= \sum_{i=0}^{d} (x_i - y_i)^2 \\ &= (x-y)\,(x-y)^T \end{aligned} \tag{23}$$

### 5.1.2. Distancia de Hamming o Cityblock

$$dist(x,y) = \sum_{i=0}^{d} |x_i - y_i| \tag{24}$$

### 5.1.3. Distancia Mahalanobis

Esta medida se diferencia de la distancia euclídea en que tiene en cuenta la correlación entre las variables aleatorias,

$$dist(x,y)^2 = (x-y)\,\Sigma^{-1}\,(x-y)^T \tag{25}$$

donde $\Sigma$ es la matriz de covariancia.

[5] Existe dos versiones de esta distancia: una, a nivel de bit; y otra, a nivel de componentes de vectores. Se utilizó la última, también conocida como ***cityblock***

## 5.2. Support Vector Machine

Las Máquinas de Vectores de Soporte (***Support Vector Machines***, o ***SVMs***) son una reciente técnica de aprendizaje supervisado desarrollado por [Vapnik 95, Vapnik 98]. SVM ha sido aplicado exitosamente a un gran número de problemas de clasificación, y es adecuado para nuestro propósito.

Como ejemplo introductorio, suponga un problema de clasificación separable en un espacio de dos dimensiones. Hay muchos hiperplanos que pueden separar las dos clases de datos, ver figura 11-(a).

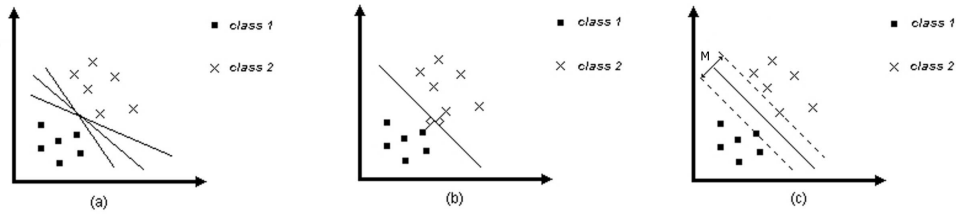



**Figura 11:** (a) Posibles hiperplanos; (b) selección de un único hiperplano que maximiza la distancia de los puntos más cercanos de cada clase; (c) hiperplano de separación óptimo que maximiza el margen.

Suponga un hiperplano tal que los puntos $x$ que se encuentren en el hiperplano satisfacen

$$\omega^T x + b = 0, \tag{26}$$

donde $\omega$ es la normal del hiperplano, $\frac{|b|}{\|\omega\|}$ es la distancia perpendicular desde el hiperplano al origen, y $\|\omega\|$ es la norma Euclidiana de $\omega$.

Definiendo ***margen*** ($M$) como la distancia entre este hiperplano y los puntos más cercanos de cada clase, figura 11-(b); el hiperplano de separación será óptimo si maximiza dicho margen, figura 11-(c).

### 5.2.1. SVM lineal

Si tenemos un conjunto de entrenamiento $\mathcal{D}$ de la forma

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^p,\ y_i \in \{-1, 1\}\}_{i=1}^n ,$$

donde $y_i$ puede ser $-1$ o $1$, indicando la clase a la cual pertenece el punto $x_i$. Se desea encontrar en (26) los $w$ y $b$ que maximicen el margen, sujeto a las siguientes restricciones,

$$\begin{aligned} \omega^T x_k - b \geq \quad 1, \qquad & \text{para} \quad y_k = 1 \\ \omega^T x_k - b \leq -1, \qquad & \text{para} \quad y_k = -1 \end{aligned}$$

Estas ecuaciones pueden combinarse con el siguiente conjunto de inecuaciones,

$$y_k(\omega^T x_k - b) - 1 \geq 0, \quad k = 1, ..., n. \tag{27}$$

Notar que si $\mathcal{D}$ es linealmente separable, se pueden seleccionar dos hiperplanos de tal manera que no haya puntos entre ellos, maximizando sus distancias, ver figura 12. Estos hiperplanos

pueden describirse como: $\omega^T x_k - b = 1$ o $\omega^T x_k - b = -1$, según la clase a la que $x_k$ corresponda. Y las distancias al origen de los hiperplanos son: $\frac{|b-1|}{\|\omega\|}$ y $\frac{|b+1|}{\|\omega\|}$, respectivamente. Entonces, el margen $M$ es igual a $\frac{2}{\|\omega\|}$ y el problema es resuelto minimizando $\|w\|$ sujeto a (27). Por conveniencia matemática, se minimiza $\frac{1}{2}\|w\|^2$ sin cambiar la solución pues el mínimo es el mismo.

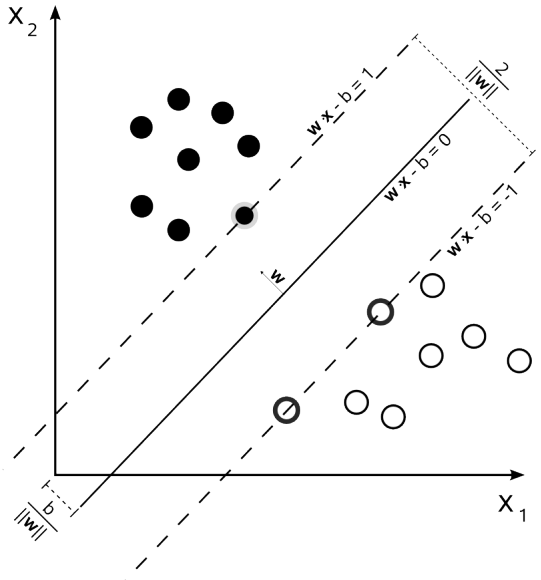



**Figura 12:** Márgenes para un SVM entrenado con datos de dos clases. Las vectores en los márgenes son llamados ***vectores soportes***

Idealmente, el modelo basado en SVM debería producir un hiperplano que separe completamente los datos en dos clases. Sin embargo, una separación perfecta no siempre es posible, ver figura 13.

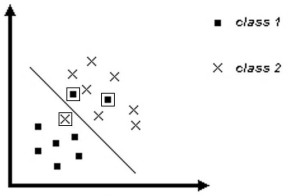



**Figura 13:** Problema de clasificación no separable

En estos casos, son añadidas $\xi_k$ (*variables slack*) en la formulación del problema con el objetivo de relajar las restricciones. Luego, el conjunto de inecuaciones toman la siguiente forma,

$$y_k(\omega^T x_k - b) \geq 1 - \xi_k, \quad k = 1, ..., n.$$

La función objetivo es entonces incrementada por una función que penaliza los $\xi_k$. La optimización se vuelve un balance entre tener el margen y la penalización de errores,

$$\begin{aligned} \underset{\omega,b,\xi}{\text{mín}} \quad & \tfrac{1}{2}\|w\|^2 + C\sum_{k=1}^{n}\xi_k \\ \text{s.t.} \quad & y_k(\omega^T x_k - b) \geq 1 - \xi_k, \quad k = 1, ..., n \\ & \xi_k \geq 0, \qquad\qquad k = 1, ..., n. \end{aligned}$$

donce $C$ es el que controla la compensación entre errores de entrenamiento y los márgenes rígidos, creando así un margen blando (*soft margin*) que permita algunos errores en la clasificación a la vez que los penaliza.

### 5.2.2. SVM no lineal - Kernels

El algoritmo resultante es esencialmente el mismo, excepto que todo producto escalar es reemplazado por una función kernel no lineal, ver figura 14.

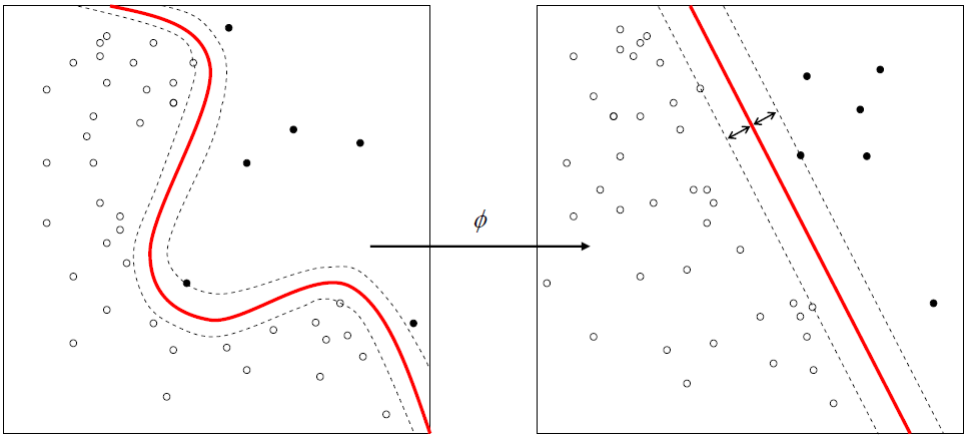



**Figura 14:** Kernel: $K(x_k, x_\ell) = \phi(x_k)^T \phi(x_\ell)$

Finalmente, el problema a optimizar es

$$\begin{array}{ll} \underset{\omega,b,\xi}{\text{mín}} & \frac{1}{2}\|w\|^2 + C\sum_{k=1}^{n}\xi_k \\ \text{s.t.} & y_k(\omega^T\phi(x_k) - b) \geq 1 - \xi_k, \quad k = 1, ..., n \\ & \xi_k \geq 0, \qquad\qquad k = 1, ..., n. \end{array}$$

y el clasificador SVM toma la siguiente forma

$$y(x) = sign\left(\sum_{k=1}^{n} \alpha_k\, y_k\, K(x, x_k) + b\right) \tag{28}$$

donde los $\alpha_k$ vienen de la solución del problema dual [Vapnik 95].

Diferentes Kernels han sido usados en la literatura pero los más populares son: Linear, Polinomial y Función de Base Radial (***Radial Basis Functions***, o ***RBF***). Definidas de la siguiente manera:

$$\begin{aligned} K_{linear}(x_k, x_\ell) &= x_k^T x_\ell, \\ K_{polinomial}(x_k, x_\ell) &= (1 + x_k^T x_\ell)^d, \\ K_{RBF}(x_k, x_\ell) &= e^{-\frac{\|x_k - x_\ell\|^2}{\gamma^2}} \end{aligned}$$

### 5.2.3. Grilla de Valores

En este trabajo los mejores resultados se consiguieron con el Kernel RBF ($K_{RBF}$), el cual posee un parámetro $\gamma$ que debe ser elegido convenientemente para cada conjunto de datos. También el parámetro $C$ debe ser escogido pues, como se indicó en la sección 5.2.1, es el que controla la compensación entre los errores de entrenamiento y los márgenes. Al ser dos parámetros, se puede generar una grilla de valores e ir probando hasta encontrar dónde se llega al máximo dentro ella. Un ejemplo de este procedimiento puede verse en la figura 15, ignorar por el momento los resultados numéricos.

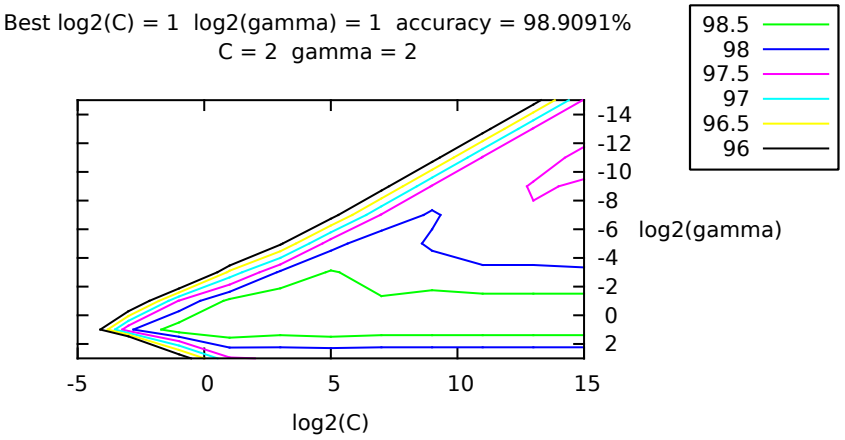



**Figura 15:** Grilla de valores de $C$ y $\gamma$, con el objetivo de encontrar los óptimos

### 5.2.4. LibSVM

Se utilizó una implementación de SVM disponible que puede encontrarse en [Chang 11].

# 6. Resultados

En esta sección se indicará sobre qué conjuntos de datos se ha trabajado y la forma de evaluación usada. También se compararán los diferentes métodos descriptos a lo largo del presente trabajo.

## 6.1. Base de datos

La base de datos utilizada se encuentra disponible en [LaViola 06]. Dicha base de datos fue subdividida en dos partes: base de datos de dígitos y base de datos de letras. La primera contiene $1100$ muestras de $11$ personas distintas que escribieron cada número $10$ veces. Y la base de datos de letras contiene cerca de $3600$ muestras. En la figura 16 se muestran $10$ ejemplos normalizados a $[0, 1]$ de la base de datos de dígitos.

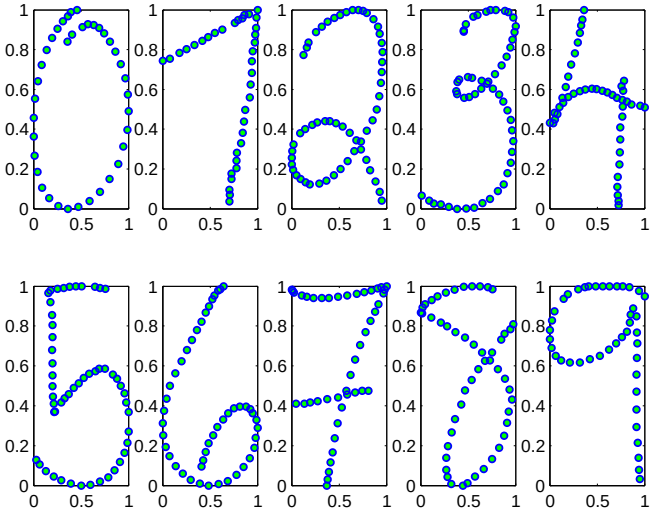


**Figura 16:** Ejemplos de la dase de datos de dígitos

Salvo que se especifique lo contrario, las gráficas que se muestran a continuación corresponden a la base de datos de dígitos, suspendiendo el análisis de los resultados para la base de datos de letras hasta la sección 6.10.

## 6.2. Validación

La validación cruzada (o ***cross-validation***), es la práctica de partir una base de datos en subconjuntos de tal forma que el entrenamiento es realizado en algunos de ellos (***training***), mientras los otros subconjuntos son retenidos para su uso posterior en la confirmación y validación del análisis inicial (***testing***).

En ***k-fold*** *cross-validation*, la base de datos es dividida en $k$ particiones. De las $k$ particiones, solo una se mantiene como datos de validación para testing, y las restante $k-1$ se utilizan para el entrenamiento. El proceso es repetido $k$ veces, de tal modo que cada una de las $k$ particiones es usada exactamente una vez para testing. Los $k$ resultados pueden ser promediados para producir una sola estimación. En este trabajo se ha utilizado 10-fold cross-validation.

## 6.3. Entendiendo las gráficas

A lo largo de esta sección se incluirán gráficas similares a la figura 17.

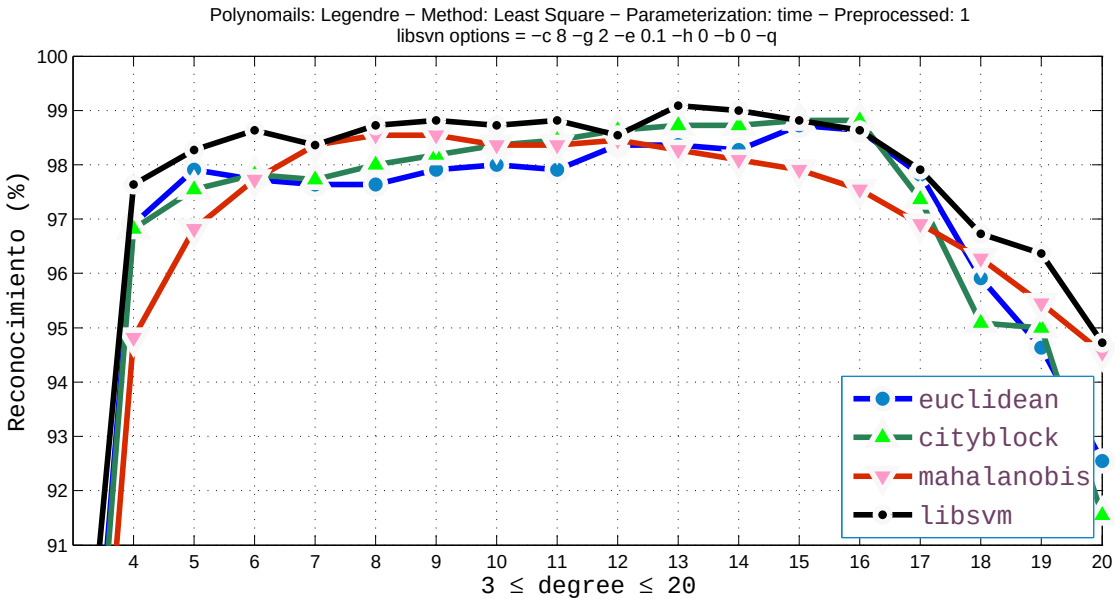



**Figura 17:** Ejemplos de gráfica.

En el título de las gráficas se indica la combinación de métodos y opciones usadas. A continuación se detallan los posibles valores:

- **Polinomio:** Legendre, Chebyshev o Legendre-Sobolev;
- **Método:** Momentos o Mínimo Cuadrados (mediante pseudo-inversa, *LeastSquare*);
- **Parametrización:** por tiempo o por longitud de arco (*arc-length*);
- **Preproceso:** activado (1) o desactivado (0);
- **libsvm:** los dos primeros parámetros son $C$ y $\gamma$, explicados en 5.2.3. Estos son pasados a la biblioteca *libsvm*. Los demás son parámetros internos de libsvm, que se han elegido convenientemente y dejado fijos en todas las pruebas.

El eje $X$ de las gráficas representan el grado $d$ (*degree*) de los polinomios elegidos, donde $d \in [3, 20]$. Recordar que los *features* son los vectores $(\alpha_0, \ldots, \alpha_d, \beta_0, \ldots, \beta_d)$ que tiene dimensión $2\,(d+1)$.

El eje $Y$ representa la ***precisión*** alcanzada por el método, porcentaje de reconocimiento correcto, en el rango $[91, 100]$.

Por último, la misma leyenda es utilizadas en todas las gráficas. Allí se indica por color los métodos de *clasificación* usados: $k$-NN, con alguna de sus distancias (Euclidiana, Cityblock o Mahalanobis), o LibSVM.

## 6.4. Momentos vs Pseudo-inversa

Se enumerarán algunas ventajas y desventajas del algoritmo de los momentos 4.3 frente a la pseudo-inversa 4.4.

**Ventajas:**

- El algoritmo se puede implementar de tal modo que los momentos se vayan calculando mientras el usuario escribe el trazo, ver 4.3.5. Por lo tanto, se puede dividir la complejidad computacional en: (1.) cálculos online (mientras se escribe), (2.) cálculos a partir de que se haya finalizado el trazo. Es necesario tomar solamente (2.) si se desea comparar con otro algoritmo que no tenga cálculos online, como el algoritmo de la pseudo-inversa. Se puede ver fácilmente que la parte (2.) del algoritmo de los momentos es **constante**. Esto sucede pues una vez calculados los momentos, solo se tiene que realizar una multiplicación matricial $\alpha = \mathbf{C}\,\mu$, para conseguir los features,
- Es más sencillo de implementar que la pseudo-inversa.

**Desventajas:**

- El algoritmo de los momentos solo funciona para los polinomios de Legendre. En cambio, con la pseudo-inversa se pueden utilizar polinomios ortogonales cualesquiera.
- El cálculo de momentos sufre de ***inestabilidad numérica*** para $d$ grandes. En la figura 18, pueden verse aproximaciones para $\{d = 9, d = 12, \ldots, d = 24\}$ de la función $u(t) = \sin(3\,\pi\,t) - \frac{1}{2}\sin\left(\frac{15\,\pi\,t}{2}\right)$.

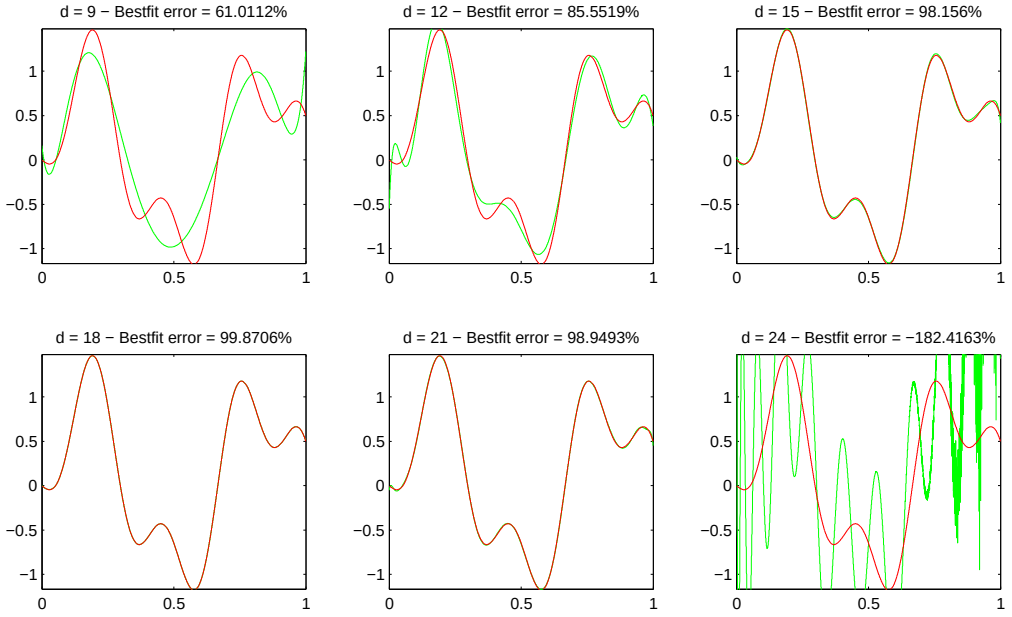



**Figura 18:** Momentos: inestabilidad numérica.

No se enumerarán las ventajas y desventajas de la pseudo-inversa frente a los momentos, pues es la relación inversa. La ventaja de un método es la desventaja del otro.

## 6.5. Preprocesar o No Preprocesar

Como se indicó en la sección 2.3.2, antes del cálculo de los features se suelen usar 3 filtros como preproceso. Estos filtros son: 1. suavizado, 2. resampling, y 3. resizing (no necesariamente en ese orden). En lo siguiente se desea examinar la posibilidad de eliminar estos pasos en pos de eficiencia.

### 6.5.1. Evitando suavizado

Recordar que los features son los coeficientes de polinomios, obtenidos por aproximación de mínimos cuadrados, lo cual genera una curva suave. El objetivo del suavizado (la eliminación del ruido presente al principio y final de cada trazo) es alcanzado sin necesidad de realizar un filtrado. Entonces, no es necesario este paso.

### 6.5.2. Evitando resampling

Al cambiar la representación de los trazos, de secuencia de puntos a curvas continuas, no hay necesidad de espaciarlos uniformemente o reducir/aumentar la cantidad de puntos en los trazos (tarea del filtro en cuestión). Si la velocidad en la que es escrito un símbolo afecta el reconocimiento, se puede optar por la reparametrización por longitud de arco, sección 4.5. Por lo tanto, tampoco es necesario el resampling.

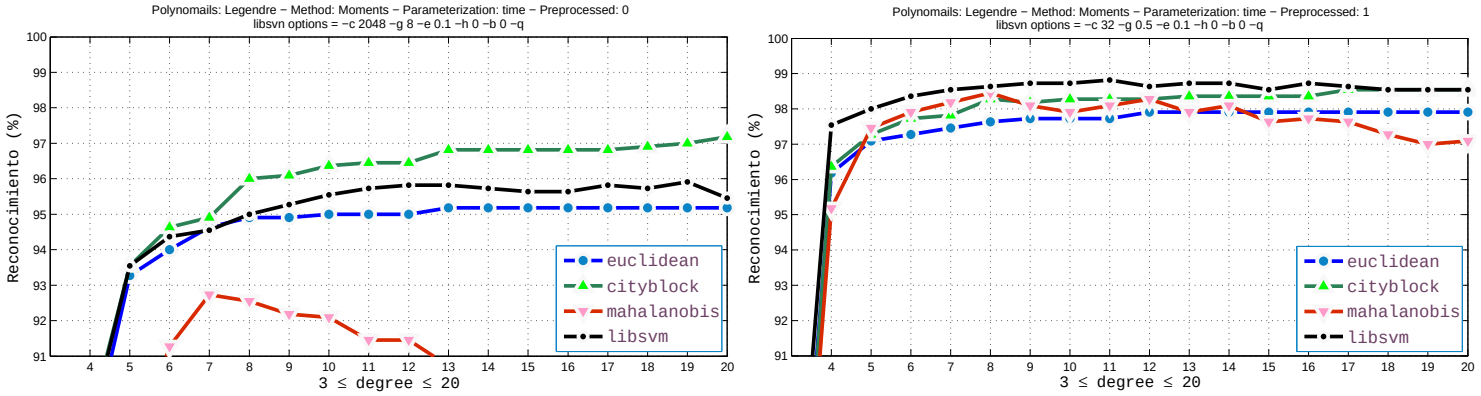
### 6.5.3. Evitando resizing

Se ha demostrado que los features obtenidos son invariantes a escala, sección 4.3.4. Por lo que también es innecesario aplicar este filtro.

### 6.5.4. Evitando traslación

Un procesamiento no mencionado es aplicar traslación, que se ocupa de trasladar el trazo de manera tal que el mínimo de cada eje sea $0$. Se ha expresado que los features no son invariantes a traslación. Aquí se muestra por experimentación, que las traslaciones no provocan una pérdida significativa en la precisión de reconocimiento. Esto permitiría tomar dos caminos posibles al implementar un sistema de reconocimiento: preprocesar o no hacerlo. Observar que a pesar de que el preproceso se haya reducido a una simple traslación, el mínimo de cada eje es conocido recién cuando el trazo es finalizado, lo cual implica que el preproceso no permite el cálculo de los momentos *mientras* se escribe el trazo, perdiendo su ventaja computacional frente al método de la pseudo-inversa.

Esta decisión de diseño puede ser tomada para el caso en que se requiera extrema eficiencia, como el caso de los dispositivos móviles, evitando preprocesar a costa de perdida precisión. Entonces, se tiene que evaluar en qué escenario se desea utilizar el sistema de reconocimiento, y elegir el balance justo entre eficiencia y precisión según sea el caso. Una comparación puede verse en la figura 19, donde al no usar preproceso en **(a)** se tiene una pérdida de precisión del $2\,\%$ (considerando los mejores resultados) con respecto a **(b)** que sí utiliza preproceso.




**(a)** Sin prepoceso **(b)** Con prepoceso

**Figura 19:** Momentos con polinomios de Legendre

## 6.6. Elección del grado $d$ de los polinomios

Recordar que llamamos $d$ al grado de los polinomios elegidos, y que los features tienen dimensión $2\,(d+1)$. Cabe señalar que a menor $d$, menos **cómputos**; pero no necesariamente a mayor $d$, más **precisión**. Se observa en las gráficas que generalmente se obtienen los máximos en el rango $[9, 15]$. Por lo que una buena elección de $d$ podría ser: $9$, $12$ ó $15$. Este es un parámetro que debe elegirse a la hora de implementar un sistema de reconocimiento usando estas técnicas.

## 6.7. Elección de la representación

La mejor representación puede variar según el escenario. Como vimos con anterioridad, la mejor representación posible teniendo como prioridad la eficiencia se logra con el método de los momentos, polinomios de Legendre (preproceso desactivado), ver figura 19a. Y la mejor representación posible en cuanto a precisión se obtuvo con los polinomios de Legendre-Sobolev (preproceso activo), ver figura 20.

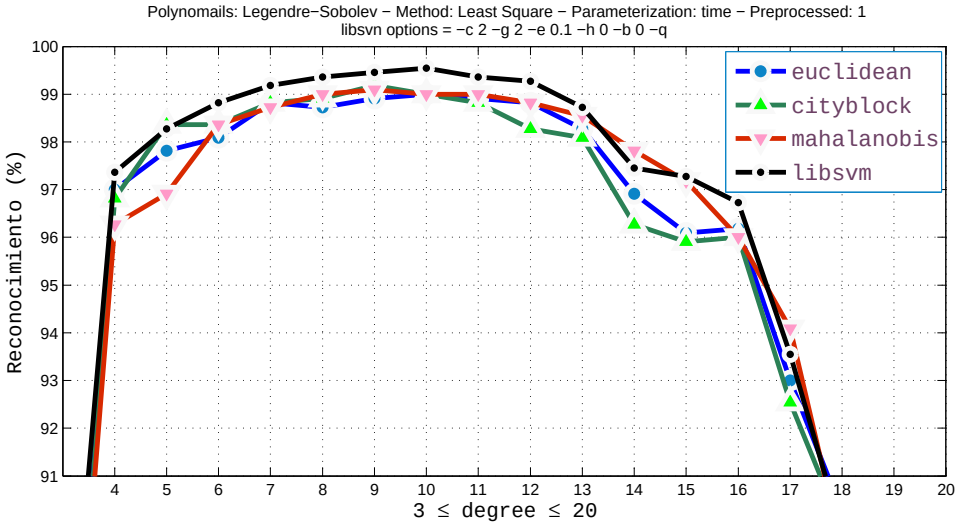
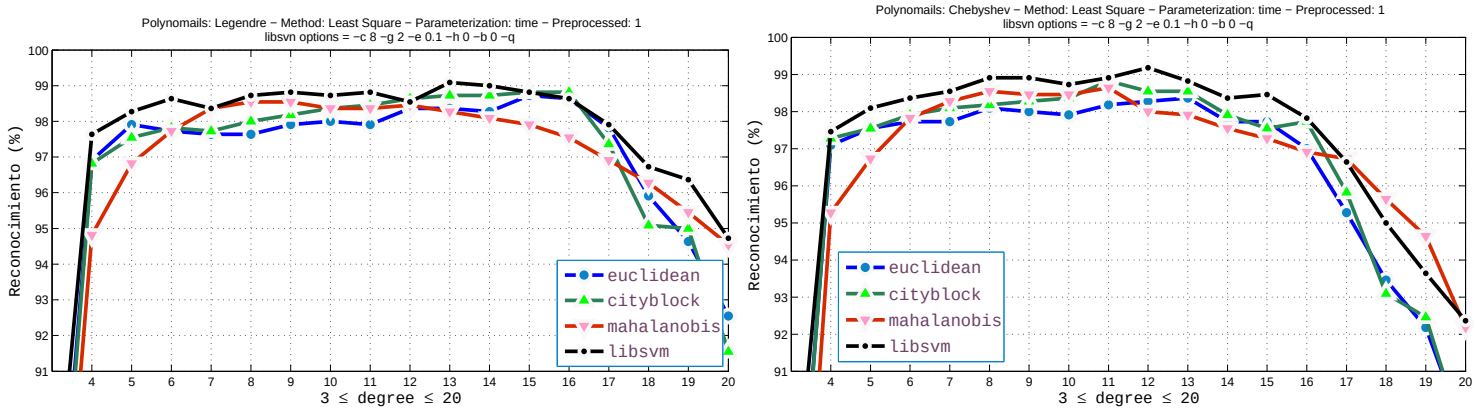



**Figura 20:** Mejor representación considerando como prioridad la precisión.

También se ha probado con los polinomios de Chebyshev (ver gráfica 21), pero no se han logrado mejoras significativas con respecto a Legendre. Motivados por la falta de ventajas, y la desventaja de no poder utilizar momentos, se ha descartado esta representación.




**(a)** Legendre **(b)** Chebyshev

**Figura 21:** Legendre vs Chebyshev

## 6.8. Parametrización por tiempo vs Parametrización por longitud de arco

Se ha indicado en 4.5 que la parametrización por longitud de arco es usualmente preferible, pues no es afectada por variaciones en la velocidad de escritura. La pruebas realizadas no indican una mejora considerable, pero se sospecha que es por la calidad que posee la base de datos, la cual no parecería tener trazos con variaciones importantes en la velocidad de escritura.

## 6.9. Momentos como features

En la ecuación (14),

$$\alpha = \mathbf{C}\,\mu \tag{29}$$

donde $C$ es la matriz de coeficientes de los polinomios de Legendre, puede verse que los únicos valores que cambian (de trazo en trazo) son los momentos $\mu$, ya que $C$ está fija (precalculada). Por qué no intentar considerar entonces a los momentos como features. En la gráfica 22 el mejor método es k-NN con la distancia de Mahalanobis. Casi que no se ven las otras medidas, por el alto rango elegido de porcentaje de reconocimiento $[91, 100]$. En estos resultados el preproceso estaba activado, pero si se desactiva el mejor método apenas supera el $90\,\%$ de precisión, con lo cual no se lo puede utilizar sin preproceso.

Observar que la mejora en la eficiencia, al evitar calcular $\alpha$, no es significativa pues se requiere una única multiplicación matricial $\mathbf{C}\,\mu$. **Es mayor la pérdida de precisión que la ganancia en eficiencia**, por lo que no se recomienda su utilización, pero sí se remarca la simpleza del método, ya que no depende de ningún polinomio.

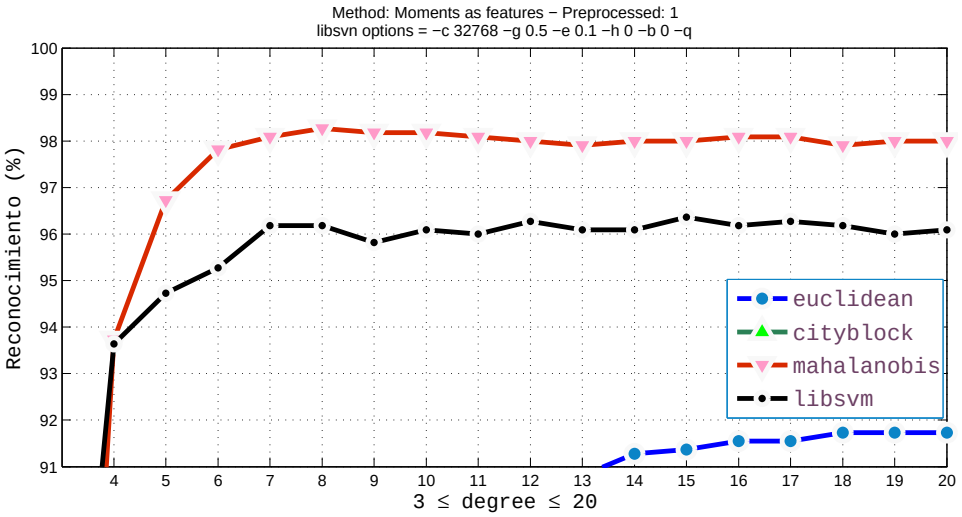



**Figura 22:** Momentos como features.

## 6.10. Features de propósitos general

Observar que los features no fueron diseñados especialmente para símbolos particulares, como ser los dígitos o las letras. Con la gráfica 23, se pretende mostrar que los features obtenidos no dependen del símbolo a ser reconocido, pues se aplicó el mismo algoritmo (sin modificaciones) a la base de datos de letras, y se consiguieron también buenos resultados. Entonces, los features se pueden considerar de propósitos generales pudiendo ser utilizados en diferentes situaciones, y no sólo con los dígitos.

En la gráfica recién mencionada, se muestran los mejores resultados en cuanto a precisión, obtenidos con los polinomios de Legendre-Sobolev de grado $9$ con poco más de $96\,\%$ de reconocimiento correcto para la base de datos de letras. Recordar que la base de datos es más grande que la de dígitos (con cerca de $3600$) y que posee más clases, por lo cual la diferencia de precisión se debe esencialmente a que la base de datos es más complicada.

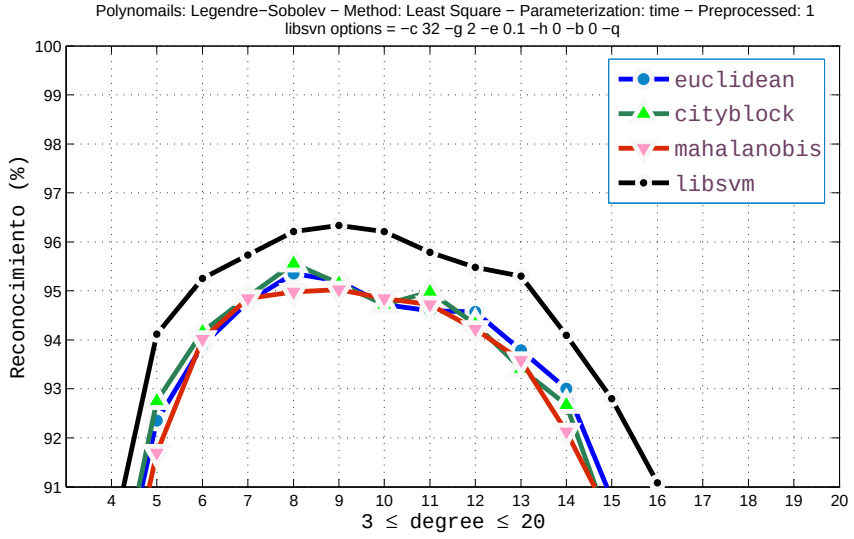



**Figura 23:** Mejores resultados para la base de datos de letras.

# 7. Posibles Aplicaciones

## 7.1. Reconocimiento de fórmulas matemáticas

Como hemos indicado anteriormente, existe una variedad importante de dispositivos electrónicos en los que se puede usar un lápiz. Sin embargo, todavía no hay una aplicación sobresaliente de reconocimiento de escritura para tales dispositivos. El usuario ve al stylus como un mouse sofisticado. Vemos en la matemática una aplicación que puede cambiar el estado actual. No sólo escribir sino también manipular expresiones matemática, obteniendo resultados inmediatos con ayuda de los poderosos sistemas de cálculo simbólico actuales, como Maple [MapleSoft 11].

Una obvia motivación es el hecho de que escribir matemática en una computadora es problemático. Por ejemplo, la expresión:

$$\int \frac{(3x^2+2)\sin(x^3+2x-1)}{\cos(x^3+2x-1)}\,dx$$

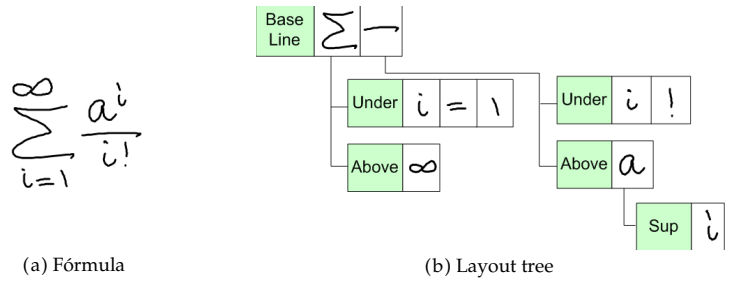
es más natural que escribir en LaTeX:

```
\int {\frac { \left( 3\,{x}^{2}+2 \right)
              \sin \left( {x}^{3}+2\,x-1 \right) }
            { \cos \left( {x}^{3}+2\,x-1 \right) }
      } ~ dx
```

En el campo de reconocimiento de escritura matemática hay un número de desafíos a sortear. A nivel de **símbolos**, existen muchos que son similares y no existe un alfabeto pequeño como lo tiene un lenguaje natural. A nivel de **entrada**, la segmentación de símbolos matemáticos es considerablemente más complicada. A nivel de construcción de **fórmulas válidas**, el texto por naturaleza es unidimensional pero expresiones matemáticas son bidimensionales, haciendo difícil determinar una línea de referencia (*baseline*) apropiada. A nivel del **renderizado**, el dibujado de símbolos matemáticos no es trivial. A nivel de **interacción**, el stylus es un dispositivo nuevo y la interacción hombre-computadora tiene que repensarse para que sea eficaz.

### Análisis estructural

Como la entrada se espera que sea matemática, además de reconocer cada caracter individualmente, también se tienen que interpretar su posición y rol en la expresión, ver figura 24.




(a) Fórmula (b) Layout tree

**Figura 24:** Representación de la fórmula manuscrita y su correspondiente Layout.

## 7.2. Reconocimiento de firmas

Un campo muy interesante, donde se podría aplicar los métodos mencionados en este trabajos, es el de verificación de firma. Pueden verse ejemplos de firma en 25. Se intentará utilizar las representaciones por polinomios ortogonales descripta, para determinar si caracterizan lo suficientemente bien a las firmas como para ser usadas en este campo.

**Figura 25:** Ejemplos de firmas

## 7.3. Reconocimiento de partituras musicales

Otra aplicación a considerar es el reconocimiento de notación musical, como puede verse en la figura 26. No se ha encontrado muchos sobre este tipo de reconocimiento usando el enfoque online.

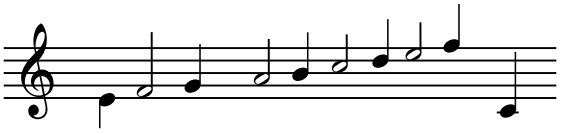


**Figura 26:** Partitura

# 8. Conclusiones

La principal motivación de este trabajo fue la intención de reconocer **fórmulas matemáticas**. Dado que el campo es complejo, el trabajo debió focalizarse en la identificación individual de símbolos, paso previo y necesario para el objetivo final de reconocimiento de expresiones matemáticas. Un requerimiento importante de los features es que sean **robustos**, en el sentido de poder usarse ya sea para dígitos, letras o símbolos matemáticos cualesquiera[6]. Lo cual se consiguió gracias a la representación utilizada.

Una observación importante es que en este trabajo solo se consideró el **primer match**, lo cual no es realista en el escenario de fórmulas matemáticas pues muchos símbolos quedan totalmente definidos dentro de un contexto. Como ejemplo, podría pensarse en una $S$ (algo más estirada) y en una $\int$ (integral), donde la desambiguación se produce cuando uno escribe los extremos de la integral o un diferencial, no quedando dudas de qué símbolo se trate.

Se logró en este trabajo una implementación muy **eficiente** del cálculo de los momentos, gracias a pensar los algoritmos como **online**, sin esperar la finalización de una etapa para comenzar la siguiente, aprovechando así los tiempos muertos que el usuario deja mientras escribe, sección 4.3.5.

También se ha mostrado que una representación con polinomios ortogonales caracteriza muy bien a los trazos, permitiendo alcanzar una alta precisión en el reconocimiento. Los resultados conseguidos con los polinomios de **Legendre-Solobev** fueron sobresalientes, acercándose al óptimo ($99{,}5\,\%$) en la base de datos de dígitos, sección 6.7.

En cuanto a la clasificación, se ha mostrado que **SVM** alcanza los mejores resultados; avalando por qué este método, gracias a su desarrollo teórico subyacente, es de los más usados actualmente en gran variedad de problemas.

En resumen, se han utilizado métodos modernos para la representación de los trazos, diferenciándose de los métodos tradicionales de reconocimiento de escritura online en los cuales los trazos son tratados como secuencias de puntos, en lugar de curvas continuas. Esta representación permite obtener excelentes resultados tanto en la **precisión** de reconocimiento como así también en la **eficiencia** computacional.

[6] Cabe destacar que posiblemente se tenga que tratar los signos de puntuación de manera discriminada por su extremada corta longitud.

## Referencias


[Abramowitz 70] **M. Abramowitz and I. Stegun.** *Handbook of mathematical functions*. 1970.

[Chang 11] **Chang, Chih-Chung and Lin, Chih-Jen.** *LIBSVM: A library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, vol. 2, pages 27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[Char 07] **Bruce W. Char and Stephen M. Watt.** *Representing and Characterizing Handwritten Mathematical Symbols through Succinct Functional Approximation*. In ICDAR, pages 1198–1202, 2007.

[Connell 00] **Scott D. Connell.** *Online Handwriting Recognition Using Multiple Pattern Class Models*, 2000.

[Golubitsky 08] **Golubitsky, Oleg and Watt, Stephen M..** *Online stroke modeling for handwriting recognition*. In Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds, CASCON '08, pages 6:72–6:80, New York, NY, USA, 2008. ACM.

[Golubitsky 10] **Golubitsky, Oleg and Mazalov, Vadim and Watt, Stephen M..** *Toward affine recognition of handwritten mathematical characters*. In Proceedings of the 9th IAPR International Workshop on Document Analysis Systems, DAS '10, pages 35–42, New York, NY, USA, 2010. ACM.

[Hausdorff 21] **Hausdorff, F..** *Summationsmethoden und Momentfolgen. I.*, 1921.

[LaViola 06] **LaViola, J.J., Jr.** *Symbol Recognition Dataset. Microsoft Center for Research on Pen-Centric Computing.* http://pen.cs.brown.edu/symbolRecognitionDataset.zip. 2006.

[MapleSoft 11] **MapleSoft.** *The Essential Tool for Mathematics and Modeling.* http://maplesoft.com, 2011.

[Nalwa 97] **Nalwa, Vishvjit S..** *Automatic On-line Signature Verification*. In Proceedings of the Third Asian Conference on Computer Vision-Volume I - Volume I, ACCV '98, pages 10–15, London, UK, 1997. Springer-Verlag.

[Talenti 87] **G Talenti.** *Recovering a function from a finite number of moments*. Inverse Problems, vol. 3, no. 3, page 501, 1987.

[Vapnik 95] **Vladimir Vapnik.** *The Nature of Statistical Learning Theory (Information Science and Statistics). ISBN 0387987800*. 1995.

[Vapnik 98] **Vladimir Vapnik.** *Statistical Learning Theory. ISBN 0471030031*. 1998.

[Watt 09] **Stephen M. Watt and Oleg Golubitsky.** *Online computation of similarity between handwritten characters*. In DRR, pages 1–10, 2009.