UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

Detección de signos de puntuación en texto manuscrito

Trabajo fin de Master

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Ignacio Pérez Muñoz

Tutor: Dr. Carlos David Martínez Hinarejos

18 de abril de 2016

A mi familia, por su gran apoyo.

Resumen

El reconocimiento de texto manuscrito (HTR, por sus siglas en inglés, *Handwriting Text Recognition*) está fundamentalmente orientado al reconocimiento de cada una de las líneas de un texto, es decir, la transcripción se realiza analizando cada una de las líneas por separado. Esto presenta el problema de que las líneas pueden recoger fragmentos de frases cuyo sentido no encaje en la estructura gramatical del idioma en que está escrito el texto, ya sea por iniciarse o finalizarse con una palabra partida como por tener puntos intermedios de fin e inicio de frase.

En este trabajo final de máster se presenta una herramienta que podría llevar a una alternativa al reconocimiento por líneas actual, cambiando las líneas que se introducen al sistema por frases completas. De este modo se espera mejorar la precisión del sistema de reconocimiento dado que la sentencia que debe reconocer sigue los modelos gramaticales de la lengua.

Con el fin de conseguir una orientación del reconocimiento por frases se hace necesario detectar los inicios y fin de frase en las líneas de texto, lo cual es equivalente a detectar signos de puntuación. En este trabajo se propone la exploración de diversas técnicas de clasificación de fragmentos de imagen de texto manuscrito, en las que se debe de detectar si se da o no un signo de puntuación en dichos fragmentos. Para ello se empleará un manuscrito en español del siglo XIX, debiendo obtenerse los fragmentos de imagen y sus etiquetas a través de reconocimiento forzado, y tras ello entrenar y probar los modelos de clasificación correspondientes.

Palabras clave: reconocimiento de texto manuscrito, htr, reconocimiento de signos de puntuación, reconocimiento de puntos, k-nn, svm

Abstract

The Handwriting Text Recognition (HTR) is principally oriented to recognize text lines, that is, the transcription is realized analyzing each different line of a text. In this framework the problem is that the lines can contain pieces of sentences whose sense does not match with the grammatical structure of the handwriting text language, either by start or end by a splitted word as to have points between the line boundaries.

In this work a new tool for HTR is presented as an alternative for current lines recognition systems, changing lines for complete sentences. In this way a better precision on the HTR system is expected because the sentence that the system has to recognize follows a regular language structure.

With the final purpose of achieving the sentence recognition, it is necessary detect first the sentence boundaries, which is equivalent to recognize punctuation. In this work a new technique for handwriting text image fragments classification is proposed, in which fragments must be identified as a punctuation mark or not. To do so, an ancient Spanish handwriting text from XIX century will be used, obtaining labeled fragments from a forced recognition. After that, the different classifiers will be trained and tested.

Keywords: handwriting hext hecogntion, htr, points recognition, k-nn, svm

Índice general

Resumen **2**

1. Introducción **6**
1.1. Estado del arte 7
1.2. Motivación 7
1.3. Objetivos 8

2. Reconocimiento de texto manuscrito **9**
2.1. Preprocesado de las imágenes 11
2.2. Modelos ocultos de Markov 12
2.3. Modelo de lenguaje 13

3. Clasificadores **15**
3.1. k vecinos más cercanos 16
3.2. Máquinas de vector soporte 17
3.2.1. Función Kernel 19

4. Extracción del corpus **20**
4.1. Corpus "Cristo Salvador" 21
4.2. Decodificación forzada 22
4.3. Normalización de imágenes 24
4.4. Extracción de fragmentos 25

5. Reconocimiento de puntos **27**
5.1. Reducción de dimensionalidad 28
5.2. k vecinos más cercanos 29
5.3. Máquinas de vector soporte 30

6. Evaluación **31**
6.1. k vecinos más cercanos 32
6.2. Máquinas de vector soporte 35

6.3. Comparativa de clasificadores 40

7. Conclusión **44**
7.1. Trabajo futuro 45

Apéndices **47**

A. Extracción del corpus **47**
A.1. iATROS 47
A.2. Gramáticas FSM 47
A.3. Principal Component Analysis 49
A.4. HTK Standard Lattice Format (SLF) 49

Índice de códigos **51**

Índice de figuras **52**

Índice de tablas **53**

Bibliografía **54**

Capítulo 1

Introducción

Vivimos en una era en la que toda nuestra información se encuentra digitalizada. Poco a poco los documentos en papel están desapareciendo y cada vez se utiliza más su versión virtual. Pocas son ya las operaciones que no podamos realizar a través de un ordenador, como la firma de documentos, pagos *online*, o la gestión de facturas entre muchas otras.

Pero dado que el ordenador es un invento moderno, son muchísimos los documentos que no están disponibles en formato digital. Tener un documento guardado en una máquina no solo nos permite poder consultarlo desde cualquier dispositivo o desde cualquier lugar, si no que las posibilidades que ofrece son enormes. A través de técnicas de análisis de datos como el *Big Data* se pueden realizar consultas a grandes bancos de información y obtener resultados en unos pocos segundos. Realizar este mismo trabajo de forma manual podría llevar días, meses o incluso años al efectuar las mismas búsquedas en libros antiguos.

Por otro lado, traspasar libros y documentos a formato electrónico resulta una tarea muy pesada, la cual supone también una gran cantidad de tiempo al tener que ir escribiendo a máquina cada uno de los libros. Para agilizar este proceso existen técnicas de reconocimiento de texto manuscrito que transcriben el texto contenido en una imagen a un texto digital, permitiendo de este modo automatizar todo el proceso de transcribir un libro o documento.

1.1. Estado del arte

Los primeros intentos para tratar de reconocer texto se llevaron a cabo tratando de realizar el reconocimiento de palabras aisladas o con un vocabulario muy restringido. Éstas primeras técnicas aparecieron para tratar de agilizar procesos administrativos, como por ejemplo reconocer las direcciones de las cartas o los datos de los extractos bancarios, y de este modo poder mejorar la rapidez con la que se gestionaban los servicios de las oficinas postales o los bancos[6].

A día de hoy, el reconocimiento de texto manuscrito está principalmente orientado a realizar un reconocimiento por líneas[11], es decir, el texto se segmenta en cada una de las líneas que lo compone y el sistema trata de transcribir las líneas una a una. Esto presenta el inconveniente de que, al realizar este tipo de reconocimiento, solo se tiene en cuenta el texto contenido en dichas líneas, pudiendo recoger éstas fragmentos de frases cuyo sentido no encaje con la gramática de la lengua en la que esté escrito el texto. Esto puede ser debido a que las líneas empiezan o terminan con una palabra que está partida o por tener signos de puntuación a mitad de una línea, consiguiendo que la sentencia que contiene dicha línea carezca de sentido al faltarle gran parte de su contexto.

1.2. Motivación

Con el fin de mejorar las técnicas de reconocimiento actuales, se sugiere una nueva alternativa a éstas. Para ello, en lugar de realizar un reconocimiento de cada una de las líneas, se propone un reconocimiento orientado a frases. Esto supone que el sistema en lugar de transcribir una única línea deberá transcribir la frase completa, y para ello deberá ser capaz de detectar cuándo comienza y termina una oración. De este modo, se trata de mitigar el error obtenido al realizar el reconocimiento de una línea cuya transcripción, al estar incompleta, no sigue las reglas gramaticales de la lengua en la que está escrito el texto.

Utilizando este nuevo enfoque de reconocimiento por frases se espera llegar a mejorar la precisión de los reconocedores de texto manuscrito, permitiendo además, seguir utilizando los mismos reconocedores que se utilizan para reconocer líneas. Esto es posible gracias a que la entrada de una frase al sistema se plantea como la entrada de una única línea, la cual es el resultado de la concatenación de todas las líneas (o fragmentos de línea) que componen

la frase.

El propósito de este trabajo será el de detectar, en una imagen que contenga texto manuscrito, el inicio y el final de cada una de las frases mediante la detección de los signos de puntuación, más específicamente los puntos. Para ello se utilizarán diversas técnicas de clasificación que permitirán detectar en una línea la posición en que se encuentran presentes cada uno de los puntos (en el caso, claro está, de que éstos se encuentren presentes en dicha línea).

1.3. Objetivos

El objetivo final de esta línea de trabajo sería conseguir unir todos los fragmentos de líneas que formen una frase completa del texto, de modo que éstos fragmentos juntos formen una única línea que contiene la toda la frase. Después se podría utilizar esta línea como entrada al sistema de reconocimiento de texto manuscrito.

Realizar toda esta tarea requiere de mucho trabajo como para llegar a realizarlo en un único trabajo final de máster. Por ello, los objetivos iniciales se han planteado como una primera aproximación en la que se realizará únicamente la detección de los signos de puntuación; más concretamente se tratará de identificar cada uno de los puntos que existan en una línea. A continuación se detallan los objetivos planteados para este trabajo:

- Obtener la posición de los puntos del corpus "Cristo Salvador" mediante reconocimiento forzado.
- Normalizar la altura de las imágenes de las líneas.
- Extraer fragmentos de las imágenes con un tamaño de ventana determinado y etiquetar como *punto* o *no punto*.
- Preparar los fragmentos en dos particiones de entrenamiento y test.
- Clasificar y evaluar utilizando distintos clasificadores.

Capítulo 2

Reconocimiento de texto manuscrito

Para comprender mejor la motivación por la cual se lleva a cabo este trabajo final de máster es necesario realizar una explicación de los conocimientos sobre los que se fundamenta. Como ya se ha comentado, la precisión de los sistemas de reconocimiento de texto manuscrito está ligado con la gramática que el sistema esté utilizando. En este capítulo se mostrará por qué ocurre esto explicando el funcionamiento de los sistemas actuales.

El reconocimiento de texto manuscrito se divide en dos tipos distintos según la cantidad de información disponible que se tiene de cada una de las muestras. Por una parte tenemos el reconocimiento *online*, en el que un usuario escribe a través de una tableta digitalizadora o un lápiz digital (Figura 2.1) mediante el cual se monitorizan los trazos que éste va realizando. De este modo no solo se obtiene la información de lo que ha escrito, si no que también de cómo lo ha escrito.

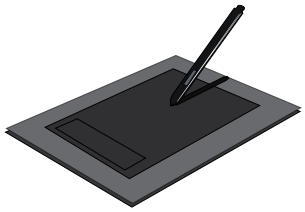


Figura 2.1: Tableta digitalizadora utilizada para el reconocimiento *online*.

Por otro lado, en el reconocimiento *offline* el sistema tan solo tiene acceso a la imagen que contiene el texto, por lo que, al no disponer de más datos, resulta más complicado obtener una transcripción. Dado que la mayoría de

las veces no se suele tener acceso a los trazos que el usuario ha realizado hasta escribir una palabra, resulta de gran importancia desarrollar buenos sistemas de reconocimiento *offline*.

El reconocimiento orientado a líneas sobre el que está orientado este trabajo forma parte del reconocimiento *offline*, por lo que durante todo este trabajo, cuando se haga relación a sistemas de reconocimiento de texto, se estará refiriendo de forma implícita a este tipo. Por ello, todo este capítulo está destinado a explicar el funcionamiento de los sistemas de reconocimiento de texto manuscrito *offline*[3, 8].

Los reconocedores de texto manuscrito modernos están basados en modelos estadísticos mediante los cuales, se busca la mayor probabilidad de que una secuencia de fragmentos, obtenidos de una imagen, correspondan a una secuencia de palabras $\hat{w}$. La ecuación fundamental viene denotada por:

$$\hat{w} = \arg\max_{w} p(w|o) \tag{2.1}$$

donde la secuencia de vectores $o = (o_1, o_2, \ldots, o_n)$ corresponden con los fragmentos obtenidos a partir de la imagen, y w es una secuencia de palabras que pertenece a un vocabulario fijo v, es decir, $w \in v^*$.

Aplicando el teorema de Bayes y asumiendo que el vocabulario es constante, la Ecuación (2.1) puede ser reescrita como:

$$\hat{w} = \arg\max_{w} p(o|w) \cdot p(w) \tag{2.2}$$

La Ecuación (2.2) muestra las dos principales incógnitas del reconocimiento de texto manuscrito. El término $p(o|w)$ es la probabilidad de que la secuencia observada o sea generada por la sentencia w. Esta probabilidad es estimada mediante Modelos Ocultos de Markov, los cuales serán explicados más adelante.

Por otra parte, $p(w)$ proporciona la probabilidad *a priori* de que una secuencia w sea escrita. Este término se estima a partir de modelos estadísticos de lenguaje, generalmente n-gramas, creados con textos de temática similar al que se quiere transcribir.

A continuación se detallan las principales partes que componen un sistema de reconocimiento de texto moderno. En primer lugar se aplica un preprocesado a las imágenes que se desean transcribir, mediante el cuál se extrae la información útil de las imágenes. Después se obtiene la transcripción utilizando el vector de características obtenido mediante el preprocesado, junto con los Modelos Ocultos de Markov y el modelo gramatical de n-gramas.

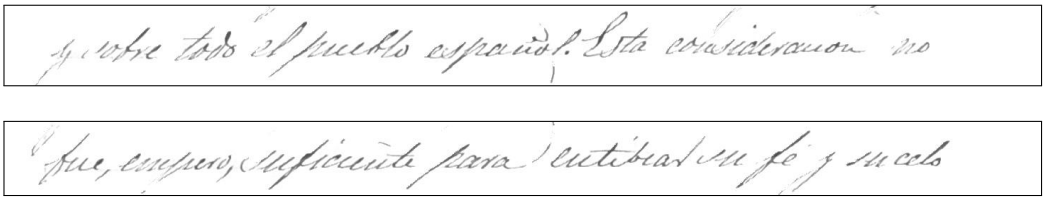


Figura 2.2: Líneas del corpus "Cristo Salvador".

2.1. Preprocesado de las imágenes

Para poder utilizar imágenes en un sistema de reconocimiento es necesario transformar estas imágenes a una estructura que el reconocedor sea capaz de manejar. Para ello se preprocesan las imágenes con el objetivo de realizar una mejor extracción de los vectores con las características principales. Los vectores de características serán más tarde utilizados por el reconocedor para obtener la transcripción. Por ello este primer paso resulta de gran importancia, dado que una mala extracción de las características o la extracción de características irrelevantes, provocará que, por muy bueno que sea el reconocedor, éste no sea capaz de clasificar correctamente.

En los documentos manuscritos se presenta el inconveniente de que, al estar escrito a mano, se producen muchas variaciones en el tamaño y forma de las letras. Además, muchos de ellos presentan manchas en el papel y variaciones en el tono de las hojas. El objetivo del preprocesado es, por tanto, obtener una imagen uniforme de la escritura, con una menor variación entre caracteres y una menor cantidad de ruido en la imagen.

El preprocesado[11] comienza con la corrección de la inclinación horizontal (*skew*) de la página. Mediante este proceso se corrige la inclinación de las líneas con respecto a la dirección horizontal. Este proceso se lleva a cabo alineando el texto de cada una de las páginas a una misma dirección.

El siguiente paso consiste en la eliminación del fondo. Mediante este proceso se eliminan las manchas que pueda contener el papel y se normaliza el color, obteniendo así una imagen en blanco y negro. Esto se consigue aplicando filtros de suavizado y realizando una normalización del color a una escala de grises.

Una vez se han nivelado las líneas y se ha normalizado el color, se procede a la división de cada una de las líneas de la imagen. Para ello se realiza una proyección de fragmentos de texto y se comprueba si existen mínimos locales que indiquen la ausencia de texto y, por tanto, la separación entre líneas.

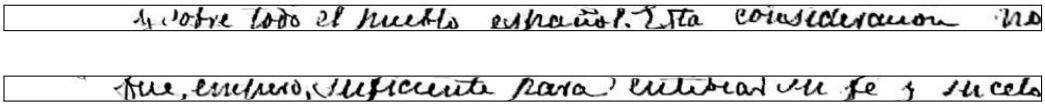


Figura 2.3: Líneas del corpus "Cristo Salvador" normalizadas.

Después se realiza una corrección de la inclinación vertical de la escritura de cada una de las líneas (*slant*), cuyo objetivo es eliminar la "cursividad" de la escritura. El último paso es aplicar una normalización del tamaño de los caracteres del texto, procurando que las letras de cada una de las líneas tengan un tamaño similar.

En la Figura 2.2 se puede ver un ejemplo de dos líneas extraídas del corpus utilizado en Capítulo 4, donde después de aplicar todo el proceso de normalizado dan como resultado las líneas mostradas en la Figura 2.3.

Finalmente, la extracción del vector de características se realiza mediante una ventana deslizante cuya altura corresponde a la altura de la imagen, y cuya anchura viene dada por un valor determinado. Dicha ventana se desliza de izquierda a derecha sobre cada una de las líneas de texto, obteniendo los niveles de grises del fragmento de la imagen contenido en su interior.

2.2. Modelos ocultos de Markov

Los modelos ocultos de Markov o HMM (por sus siglas del inglés, Hidden Markov Models), se utilizan para definir el modelo óptico del sistema de reconocimiento de texto manuscrito, y son ampliamente utilizados en el campo del reconocimiento de patrones. Su aplicación original estaba destinada al reconocimiento de audio, pero debido a las similitudes que ambos problemas presentan, los HMM se han vuelto también muy populares en el reconocimiento de texto manuscrito. Su principal ventaja es que permiten el reconocimiento de caracteres incluso cuando éstos presentan deformaciones debidas al alargamiento de la escritura.

Un modelo oculto de Markov es un autómata finito estocástico en el que cada una de las transiciones tiene asociada una probabilidad conocida, y cada uno de los estados tiene asociado la probabilidad de generar una serie de símbolos (Figura 2.4).

En el reconocimiento del texto manuscrito, se asume que los vectores de características obtenidos de la secuencia de fragmentos observada a partir de la imagen de una línea se corresponde con un modelo oculto de Markov.

Éstos son utilizados para determinar cada uno de los caracteres y la unión de varios modelos ocultos de Markov permite identificar las palabras.

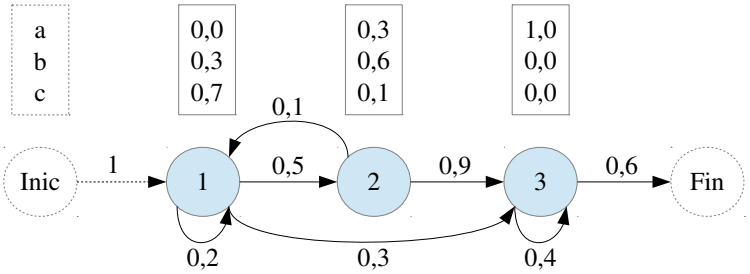


Figura 2.4: Ejemplo de un Modelo Oculto de Markov discreto.

Para reducir el espacio de búsqueda de las palabras, los modelos ocultos de Markov trabajan junto con un modelo léxico que proporciona información sobre la estructura de cada una de las palabras que forman el vocabulario. Así pues, estos modelos ayudan a determinar qué secuencia de fragmentos de una línea se corresponde con cada palabra.

2.3. Modelo de lenguaje

Los modelos de lenguaje definen la estructura gramatical de una lengua. Mediante estos modelos se guía al reconocedor para que proporcione una transcripción acorde al contexto de un idioma.

En sistemas de reconocimiento de habla y texto, los modelos de lenguaje están compuestos a partir de las distintas palabras del vocabulario. Como se ya se ha comentado, el término $p(w)$ de la Ecuación (2.2) equivale a la probabilidad a priori de que la secuencia de palabras w pueda ser escrita. Si w está compuesta por n palabras, y el tamaño del diccionario a utilizar es $|v|$, el número de posibles secuencias de palabras equivale a $|v|^n$. Incluso con un tamaño de vocabulario $|v|$ reducido, el espacio de búsqueda es muy grande, por lo que es necesario hacer uso de modelos estadísticos del lenguaje para reducir la búsqueda.

Para poder disminuir el espacio de búsqueda de los modelos de lenguaje se utilizan modelos de n-gramas[2] con un número de n no muy grande (normalmente entre 2 y 3), los cuales tienen asociados la probabilidad de que se produzca cada uno de los n-grama. La obtención del vocabulario y las probabilidades de los n-gramas se realiza utilizando grandes volúmenes de texto

de temática similar a los que se desea reconocer para que posteriormente la transcripción sea lo más precisa posible.

Existen también otros modelos estadísticos de lenguaje utilizados para representar gramáticas, como las gramáticas basadas en máquinas de estados finitas utilizadas en el Capítulo 4.

Capítulo 3

Clasificadores

Un sistema de reconocimiento se reduce a un clasificador $\hat{f}(x)$ que determina la clase a la que pertenece el objeto x por el cual se le está preguntando. El número de clases $\mathbb{C}$ se establece según la cantidad de objetos distintos que se quieran reconocer. En este caso, se desea detectar cuándo un fragmento de imagen x contiene un punto o no, por lo que nos encontramos con un problema de clasificación de dos clases $C = \{0, 1\}$ en el que si una imagen contiene un punto se etiqueta como 1, y si no lo contiene se etiqueta como 0. Es decir $\hat{f}(x) \in C$ y $\mathbb{C} = 2$.

El reconocimiento de puntos entraría dentro de los clasificadores que utilizan una técnica de aprendizaje denominada como *aprendizaje supervisado*. Esta técnica se caracteriza por elaborar una función matemática a partir de muestras de entrenamiento previamente etiquetadas. De esta manera, el clasificador aprenderá a etiquetar cada una de las muestras, pero antes se requiere un proceso de aprendizaje.

Aunque existen una gran multitud de métodos de clasificación mediante aprendizaje supervisado, en este capítulo solo se expone el funcionamiento de los clasificadores que más tarde serán utilizados en el reconocimiento de puntos. En el Capítulo 5 se puede ver más acerca de la implementación de estos clasificadores, mientras que en el Capítulo 6 se exponen los resultados obtenidos con cada uno de éstos clasificadores.

3.1. k vecinos más cercanos

El clasificador por los k vecinos más cercanos o k-NN (por su siglas del inglés, *k-Nearest Neighbors*)[10] es uno de los clasificadores más sencillos. Se basa principalmente en la idea de suponer que los vectores de características de las muestras pertenecientes a una misma clase se encontrarán en un espacio vectorial cercano. El clasificador k-NN se define formalmente por las siguientes fórmulas:

$$\hat{f}(x) = \arg\max_{c \in C} \sum_{i=1}^{k} \delta(c, f(x_i)) \tag{3.1}$$

$$\delta(a, b) = \begin{cases} 1 & \text{Si } a = b \\ 0 & \text{Otro caso} \end{cases} \tag{3.2}$$

Su funcionamiento consiste en comprobar la distancia del vector de características de la muestra a clasificar x contra un conjunto de muestras previamente clasificadas. Después se escogen las k muestras más cercanas a la muestra a clasificar $(x_1, x_2, ..., x_k)$, y se comprueba cuál es la clase de las muestras que más se repite tal y como se puede ver en la Ecuación (3.1). Para comprobar la distancia de los vectores se utiliza generalmente la distancia euclídea:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p} (x_{ri} - x_{rj})^2} \tag{3.3}$$

donde p corresponde al número de dimensiones de los vectores de características.

En la Figura 3.1 se muestra un ejemplo de clasificación para un vector de características de dos dimensiones y dos clases. Como se puede apreciar en el dibujo, tomando como k los tres vecinos más cercanos el clasificador etiquetaría la muestra verde con la etiqueta de la clase de los triángulos rojos. Sin embargo, tomando $k = 5$ aparecen más muestras de la clase contraría por lo que ahora el clasificador clasifica la muestra con la misma etiqueta que el cuadrado azul.

Este clasificador es muy sencillo, pero tiene el inconveniente de que requiere una gran cantidad de memoria a la hora de clasificar, y conlleva un tiempo computación alto, dado que tiene que tener almacenado todo el con-

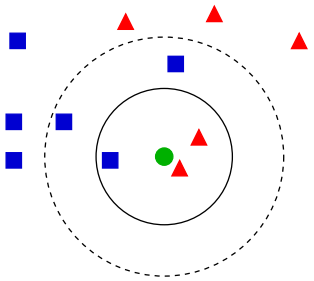


Figura 3.1: Ejemplo de un clasificador k-NN. Si $k = 3$ la muestra pertenece a los triángulos, si $k = 5$ se corresponde con un cuadrado.

junto de entrenamiento para poder realizar posteriormente la comparación de las distancias y elegir a los vecinos más cercanos.

Existen diversas variantes de este clasificador cambiando la función de distancia, como por ejemplo utilizando distancia de Mahalanobis, o ponderando la contribución de la clase de cada vecino dando mayor importancia a la clase de los vecinos que se encuentren más cercanos.

3.2. Máquinas de vector soporte

Las máquinas de vector soporte o SVM (por sus siglas en inglés, *Support Vector Machines*) son otro método de clasificación mediante aprendizaje supervisado. Realizan una clasificación a través de la introducción de hiperplanos de decisión. Cada uno de estos hiperplanos realiza una división del espacio vectorial en dos semiespacios, de modo que las muestras quedan divididas en el espacio según su etiqueta de clase.

En un problema de clasificación de C clases, cada hiperplano de decisión queda definido por una función discriminante lineal generalizada ϕ_c con $1 <= c <= C$. Esta función es capaz de clasificar una muestra x con la ayuda de un modelo de parámetros Θ, el cual está compuesto por una serie de vectores de pesos $\theta_c \in \mathbb{R}^d$ y sus respectivos umbrales $\theta_{c0} \in \mathbb{R}$.

$$\phi_c : \mathbb{R}^d \to \mathbb{R} : \phi_c(x, \Theta) = \theta_c^t x + \theta_{c0} = \sum_{i=1}^{d} \theta_{ci}^t x_i + \theta_{c0} \tag{3.4}$$

Haciendo uso de la Ecuación (3.4) el clasificador f es capaz de clasificar una muestra x gracias a la regla de clasificación de la Ecuación (3.5), donde se comprueba cual es la clase que devuelve un valor máximo en la función

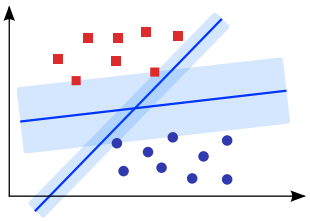


Figura 3.2: Ejemplo de dos hiperplanos donde se puede ver en azul claro el margen con respecto a los vectores soporte.

discriminante de cada clase.

$$f(x) \stackrel{\text{def}}{=} \hat{c} = \underset{1 \leq c \leq C}{\arg\max}\, \phi_c(x, \Theta) \iff \phi_{\hat{c}}(x, \Theta) > \phi_c(x, \Theta) \quad \forall c \neq \hat{c} \tag{3.5}$$

El modelo de parámetros Θ se estima a través de un conjunto de muestras etiquetadas S mediante un proceso de aprendizaje. En este proceso de aprendizaje se busca maximizar la distancia entre los vectores de cada clase con el hiperplano de decisión. Los vectores situados en las líneas paralelas al hiperplano se denominan *vectores soporte*. En la Figura 3.2 se pueden ver dos hiperplanos de decisión junto con el margen a los vectores soporte de cada clase.

Existen múltiples maneras de definir un hiperplano clasificador mediante valores diferentes de θ_c y θ_{c0}. Mediante el proceso de aprendizaje se pretende obtener el hiperplano cuyo margen con respecto a los vectores soporte x^* sea máximo. Por ello a los clasificadores SVM se les conoce también como *clasificadores de margen máximo*. La distancia r^* del vector $x^* \in S$ con respecto al hiperplano se calcula como:

$$r^* = \frac{|\theta_c^t\, x^* + \theta_{c0}|}{\|\theta_c\|} = \frac{1}{\|\theta_c\|} \tag{3.6}$$

por lo que el margen de H con respecto al conjunto de muestras de entrenamiento S queda definido como:

$$2r^* = \frac{2}{\|\theta_c\|} \tag{3.7}$$

La obtención del margen máximo se calcula mediante algoritmos de optimización. En las SVM se utiliza la técnica de los multiplicadores de Lagrange, donde en lugar de buscar la distancia máxima de $2r^*$, se realiza

una búsqueda de los parámetros que minimicen $\frac{1}{2}\theta_c^t\theta_{c0}$. Al desarrollar la función de Langrange se obtiene la Ecuación (3.8), donde α_n^* corresponde con los multiplicadores de Lagrange óptimos, c_n es la clase de la muestra n, y $\nu = \{n \in \mathbb{N}, 1 \leq n \leq N \,|\, \alpha_n^* \neq 0\}$, donde N es el número de clases.

$$\phi_{svm}(x, \Theta) = \sum_{n \in \nu} \alpha_n^* c_n x_n^t x + \theta_0^* \tag{3.8}$$

3.2.1. Función Kernel

Debido a que mediante los multiplicadores de Lagrange los parámetros θ y θ_0 se obtienen en base a productos escalares entre muestras de entrenamiento, podemos definir funciones Núcleo o Kernel (por su traducción al inglés) mediante las cuales se reemplaza la combinación lineal de las muestras mediante una nueva función. Gracias a la introducción de las funciones Kernel se consigue reducir el coste computacional del algoritmo de optimización. Al aplicar las funciones Kernel en la Ecuación (3.8) se cambia el producto lineal $x_n^t x$ por la nueva función $K(x_n, x)$, dando como resultado la Ecuación (3.9):

$$\phi_K(x, \Theta) = \sum_{n \in \nu} \alpha_n^* c_n K(x_n, x) + \theta_0^* \tag{3.9}$$

En la Tabla 3.1 se puede ver un ejemplo de algunas de las funciones Kernel más utilizadas.

Lineal	$K(x_i, x_j) = x_i \cdot x_j$
Polinomial	$K(x_i, x_j) = (x_i \cdot x_j)^2$
Perceptrón	$K(x_i, x_j) = \|x_i - x_j\|$
Gaussiano	$K(x_i, x_j) = e^{-\frac{(x_i - x_j)^2}{2 \cdot \sigma^2}}$
Sigmoid	$K(x_i, x_j) = \tanh(x_i \cdot x_j - \theta)$
Radial	$K(x_i, x_j) = e^{-\gamma \cdot \lvert x_i - x_j \rvert^2}$

Tabla 3.1: Funciones Kernel utilizadas en SVM.

Capítulo 4

Extracción del corpus

A la hora de realizar un reconocedor es necesario disponer de un corpus con el que poder entrenar y evaluar los algoritmos que lo componen. Dicho corpus tiene que estar formado por muestras similares a las que el reconocedor tendrá que enfrentarse a la hora de ser utilizado en un entorno real. Por ello, cuantas más muestras dispongamos mejor se comportará nuestro clasificador, y podremos obtener unas estadísticas de fiabilidad que se acerquen más a lo que sería su comportamiento en un entorno final.

Durante este capítulo se detalla todo el procedimiento que se ha seguido hasta obtener el corpus que, más adelante, será utilizado para comprobar la capacidad de los clasificadores. Dado que lo que se quiere detectar son puntos en las líneas, para este trabajo es necesario obtener imágenes de *puntos* y *no puntos* con los que poder entrenar el clasificador. De este modo, éste debe de ser capaz de, dada una imagen, distinguir si se trata de un punto o no. Por ello, lo primero que se necesita son fragmentos de imágenes que contengan puntos y fragmentos de imágenes que no los contengan.

Debido a que se requiere una gran cantidad de imágenes de puntos y no puntos, es necesario automatizar el proceso de extracción de las muestras. Para poder llevar a cabo dicha automatización se han creado diversas herramientas que facilitan el trabajo. Principalmente se han utilizado *scripts* para la consola de *Linux*, debido a su facilidad para realizar acciones repetitivas en grandes bloques de ficheros. También se han creado varios *scripts* en lenguaje *Python* para las tareas relacionadas con los modelos de lenguaje, pues resulta más versátil a la hora de manejar diccionarios de palabras.

En las siguientes secciones se detallan cada uno de los pasos que se han seguido hasta obtener el corpus final con el que posteriormente se entrenarán

los clasificadores.

4.1. Corpus "Cristo Salvador"

Las imágenes de puntos y no puntos se han obtenido a partir de un corpus ya existente llamado "Cristo Salvador". Este corpus ha sido utilizado por el centro de investigación *Pattern Recognition and Human Language Technology* para investigar en áreas de reconocimiento de texto manuscrito[9].

Este corpus esta compuesto por un conjunto de páginas escaneadas de un manuscrito del siglo XIX titulado *Noticia histórica de las fiestas con que Valencia celebró el siglo sexto de la venida a esta capital de la milagrosa imagen del Salvador*[1], el cual está escrito por Vicente Boix en 1853.

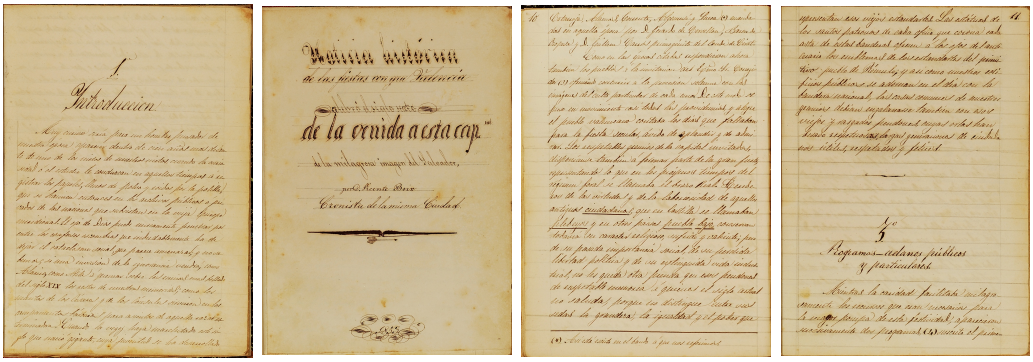


Figura 4.1: Páginas extraidas del corpus "Cristo Salvador".

En total el corpus contiene 53 hojas escaneadas a color a una resolución de 300 dpi. Todas las páginas han sido escritas a mano por un único autor y en ellas se pueden apreciar manchas en el papel, variaciones en el tono del fondo, distintos niveles de iluminación y manchas debidas a la humedad y a derrames de tinta. Además, el texto contiene palabras con distintos tamaños, y partes del texto que se encuentran subrayadas, como se puede apreciar en las páginas de la Figura 4.1.

Para extraer el corpus de puntos más cómodamente se ha utilizado una versión del corpus en el que las imágenes están ya preprocesadas y normalizadas en blanco y negro. Así mismo, las imágenes de las páginas han sido previamente divididas por líneas, obteniendo por tanto imágenes similares a las de la Figura 4.2. El preprocesado que se ha aplicado en la extracción de las líneas es el mismo que se detalla en el Capítulo 2.

El corpus final utilizado está compuesto por un total de 1214 imágenes

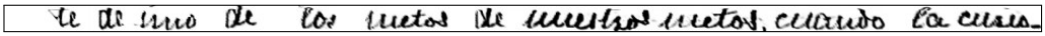


Figura 4.2: Ejemplo de una linea del corpus "Cristo Salvador" ya normalizada en la que se puede leer la frase "*te de uno de los nietos de nuestros nietos, cuando la curio-*".

Nº de páginas	53
Nº de líneas	1.214
Nº de palabras	11.078
Nº de caracteres	63.107

Tabla 4.1: Datos sobre el corpus "Cristo Salvador".

de líneas, las cuales contienen 314 puntos repartidos entre 297 imágenes de dichas líneas. Además, las imágenes varían tanto en altura como en anchura. Se debe comentar que para cada una de las líneas del corpus se tiene además su transcripción, su vector de características y un HMM ya entrenado para dicho corpus. La Tabla 4.1 resume las magnitudes más relevantes del corpus.

La elección de este corpus, además del hecho de que ya se tuvieran las líneas extraídas, viene dada por el hecho de que es un corpus que ya ha sido utilizado para probar un sistema de reconocimiento de texto manuscrito, y por tanto ya se tienen resultados sobre su precisión. Esto permitirá en trabajos futuros realizar comparaciones y ver si realmente el reconocimiento orientado a frases mejora o no el reconocimiento por líneas.

Cabe decir que algunas de las transcripciones del corpus utilizado no son del todo correctas, dado que se han encontrado algunos errores de transcripción en los que el texto no correspondía totalmente con el texto manuscrito de la imagen. Se pueden encontrar algunos ejemplos en la Figura 4.3.

En los siguientes apartados se detalla el procedimiento que se ha seguido hasta obtener el corpus final de *puntos* y *no puntos*. Dicho corpus se utilizará en el Capítulo 6 para comprobar la evaluación de los clasificadores detallados en el capítulo anterior.

4.2. Decodificación forzada

El primer paso para obtener los puntos es detectar su posición dentro de las imágenes de las líneas, para lo que hace falta saber el píxel en el que empieza el punto y el píxel en el que éste acaba.

Para poder detectar dichas posiciones se ha utilizado el programa de reconocimiento de texto manuscrito iATROS (Apéndice A.1), con el cual se

ha realizado una decodificación forzada de la imagen, es decir, se obliga al programa a reconocer únicamente la transcripción de la línea de la cual se quieren extraer las posiciones de los puntos. Este tipo de reconocimiento se consigue generando una gramática en la que la única sentencia que sea capaz de generar sea la transcripción de la imagen. La gramática utilizada en este caso tiene un formato diferente al habitual, dado que no se utilizan n-gramas, si no que se utilizan gramáticas formadas por máquinas de estados finitos o FSM (por sus siglas en inglés, *Finite State Machines*) (Apéndice A.2).

Para empezar, se ha generado una gramática para cada una de las líneas a partir del su transcripción. También se ha creado un fichero con el nuevo modelo léxico, el cual se ha creado a partir del vocabulario de todas las palabras contenidas en todas las gramáticas.

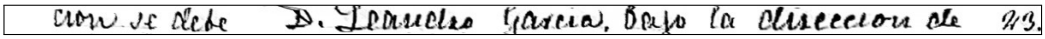
La decodificación forzada ha supuesto una de las tareas más complejas de este trabajo. Para empezar, los vectores de características de cada una de las imágenes no correspondían con el número de columnas que cada imagen presentaba. Por ello ha sido necesario volver a generar los ficheros con los vectores de características, de modo que cada fichero contuviesen tantos vectores como columnas tuviese su imagen asociada.

Una vez realizado el reconocimiento forzado, la posición de los puntos de cada imagen viene dado en cada uno de los ficheros de *lattice* asociado a dicha imagen. Estos ficheros de *lattice* son generados por iATROS y están escritos en formato SLF (Apéndice A.4), el cual ha sido desarrollado por el equipo de HTK[1][5]. Entre otros datos, cada fichero contiene las palabras reconocidas y el intervalo de columnas en el que iATROS predice que éstas palabras están representadas.

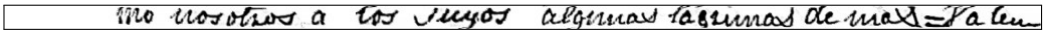
A pesar del reconocimiento forzado, no se ha conseguido que todas las posiciones sean correctas, por lo que ha sido necesario una inspección manual para revisar los puntos mal situados y corregir con la posición correcta. También se han producido errores debido a que en el corpus hay líneas cuya transcripción no es correcta.

En la Figura 4.3 se muestran tres ejemplos de imágenes de algunos de los errores que se han detectado en el corpus “Cristo Salvador”. En la imagen (a) aparece un punto después de la “D” que no queda reflejado por la transcripción. La omisión del punto en las transcripciones es bastante habitual encontrarlo en el tratamiento. La imagen (b) muestra un ejemplo donde el punto no aparece debido al símbolo “=”. El último ejemplo es la imagen (c),

[1]Herramienta para la creación y manipulación de HMMs, utilizada en multitud de sistemas de reconocimiento.



(a) *"cion se debe a D Leandro Garcia, bajo la direccion de 23."*



(b) *"mo nosotros, a los suyos algunas lagrimas de mas. = Valen-"*

(c) *"con aquella, de la redaccion del programa."*

Figura 4.3: Líneas con errores de transcripción.

donde la imagen no está normalizada y la línea aparece cortada.

4.3. Normalización de imágenes

Los vectores de características se extraen de los fragmentos de imágenes de puntos y no puntos, y están formados directamente por el nivel de gris de cada uno de los píxeles de los fragmentos. El nivel de gris se mide como un número entero comprendido entre 0 y 255, donde el 0 es el negro y 255 equivale al blanco. Los vectores se consiguen al redimensionar la matriz de grises de fragmentos de imagen de tamaño $m \times n$, en un único vector de tamaño $1 \times nm$.

Dado que cada una de las imágenes de las líneas tiene un tamaño diferente, es necesario normalizar la altura de éstas para poder extraer vectores de características del mismo tamaño. En el caso de la anchura, no será necesario realizar el normalizado dado que posteriormente se extraen los vectores de características utilizando una ventana deslizante de tamaño fijo.

En una primera instancia se optó por un normalizado de todas las líneas hasta igualar la altura de cada imagen a la altura de la línea de altura máxima, que en este caso tiene una altura de 123 píxeles. En la Figura 4.4 se muestra en ejemplo de la imagen de máxima altura junto con la imagen normalizada a dicha altura.

Debido a errores de extracción de líneas, o a que algunas de las líneas contienen títulos de páginas o capítulos con una escritura distinta a la normal, se ha decidido eliminar todas aquellas líneas cuya altura fuera manifiestamente superior o inferior a la altura media. De este modo se han descartado todas las líneas inadecuadas que difieren de lo que podría considerarse una línea normal del corpus, es decir, se han desechado los valores atípicos (en inglés,

outliers), dado que el uso de dichas líneas podrían afectar negativamente al clasificador. Después de realizar el cálculo de los valores atípicos se han seleccionado tan solo aquellas imágenes de líneas cuya altura esté comprendida entre 20 y 34 píxeles. Finalmente se han obtenido un total de 1118 líneas, las cuáles contiene 260 puntos repartidos entre 249 de las líneas.

El normalizado de las líneas se realiza mediante la agregación de píxeles en blanco en la parte superior e inferior de la imagen hasta igualar la altura de la imagen con la altura máxima. En este caso la línea de mayor altura seleccionada contiene 34 píxeles, por lo que todas las demás líneas se han normalizado a dicha altura.

Figura 4.4: Línea de altura máxima junto con una línea normalizada a la misma altura.

4.4. Extracción de fragmentos

El último paso para obtener el corpus de puntos es realizar la extracción de fragmentos de las líneas, etiquetando cada uno de estos fragmentos como *punto* o *no punto* cuando proceda. La extracción de fragmentos se realiza mediante una ventana deslizante de un tamaño determinado a lo largo de todas las imágenes de líneas. El desplazamiento utilizado para mover la ventana ha sido de una columna, dando como resultado la generación de más de 1.000.000 de fragmentos de imágenes para cada uno de los tamaños de ventana.

Cada vez que un fragmento es extraído es etiquetado según la clase que le corresponda. Para ello se revisa si la posición del fragmento en la línea coincide con las posiciones obtenidas mediante la decodificación forzada. Se han tenido en cuenta como punto todos aquellos fragmentos que contuvieran una parte, por pequeña que fuese, del punto.

Para probar los clasificadores se han extraído fragmentos de ventanas de 15 tamaños distintos, utilizando ventanas con un ancho de 5, 7, 10, 12, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90 y 100 píxeles. En la Tabla 4.2 se muestra la cantidad de fragmentos que se han obtenido en cada caso. Como puede apreciarse, gracias a la ventana deslizante se consigue una cantidad enorme

de fragmentos, obteniendo finalmente más de 15 millones de fragmentos entre todos los tamaños de ventana utilizados.

Tam. Ventana	Puntos	No puntos
5	5.003	1.164.131
7	5.432	1.161.466
10	6.067	1.157.477
12	6.489	1.154.819
15	7.120	1.150.834
20	8.170	1.144.194
25	9.220	1.137.554
30	10.270	1.130.914
40	12.358	1.117.646
50	14.421	1.104.403
60	16.461	1.091.183
70	18.474	1.077.990
80	20.445	1.064.839
90	22.362	1.051.742
100	24.230	1.038.694

Tabla 4.2: Número de fragmentos obtenidos con distintos tamaños de ventana

Capítulo 5

Reconocimiento de puntos

El reconocimiento de puntos se plantea como un problema de clasificación de dos clases, donde las muestras son etiquetadas como *punto* o como *no punto*. Durante este capítulo se detalla la implementación de cada uno de los clasificadores y los problemas que han surgido durante su desarrollo.

Debido al elevado volumen de datos, durante todo el desarrollo de este trabajo han aparecido una gran cantidad de problemas relacionados con la memoria. Mientras se realizaba la ejecución de muchos de los algoritmos utilizados el ordenador se ha quedado sin memoria, produciendo numerosos errores y cierres inesperados de aplicaciones, resultando este problema un gran quebradero de cabeza.

El lenguaje de programación escogido para la implementación de los algoritmos de clasificación ha sido *Octave*/*Matlab*, ya que estos programas incluyen paquetes de funciones enfocados al cálculo estadístico y al tratamiento de imágenes, por lo que a la hora de desarrollar un algoritmo agilizaban bastante el trabajo a realizar. También comentar que inicialmente se empezaron a programar los clasificadores utilizando únicamente *Octave*, ya que al tratarse de una herramienta *Open Source*, se puede disponer de ella más fácilmente. Más tarde, debido a la capacidad de *Matlab* de paralelizar la ejecución de ciertas funciones, se tomó la decisión de cambiar de entorno y ejecutar todos los algoritmos bajo *Matlab*, debido a que el tiempo de ejecución comenzó a ser también un inconveniente bastante serio.

El primer impedimento que se ha encontrado ha sido el hecho de que en *Octave* el tamaño máximo de memoria por defecto destinado a *arrays* de datos es de 2GB; esto es debido a que, por defecto, este programa tan solo utiliza variables de 32 bits. A pesar de que las imágenes utilizadas no tienen

	Puntos (20 %)	No puntos (80 %)	Total
Entrenamiento	3.500	14.000	17.500
Evaluación	1.500	6.000	7.500

Tabla 5.1: Distribución de muestras de los conjuntos de entrenamiento y evaluación

una gran resolución, el hecho de que se disponga de una cantidad tan grande de datos ha dado como resultado que las variables utilizadas en el código sean muy grandes. Durante la ejecución de los algoritmos *Octave* daba errores de memoria continuamente, por lo que, para solucionar este problema, se ha tenido que recompilar de nuevo *Octave* junto con todas las dependencias que este programa utiliza para que, de este modo, *Octave* pueda funcionar con variables de 64 bits y acepte *arrays* de tamaño mucho mayor.

A pesar de haber solucionado esta cuestión, los problemas de memoria han seguido apareciendo, produciendo que *Octave* y *Matlab* siguiesen dando errores de ejecución o cerrándose de forma inesperada. Además, los tiempos de cálculo para las muestras obtenidas para cada tamaño de ventana estaban siendo muy elevados, por lo que al final se ha optado por una reducción considerable del corpus utilizado.

Para cada uno de los tamaños de ventana se ha seleccionado aleatoriamente un subconjunto de datos formado por 5.000 *puntos* y 20.000 *no puntos*, quedando así una proporción de un 20 % de *puntos* y un 80 % de *no puntos* del total de muestras.

Además, el corpus se ha dividido en dos partes, utilizando una parte para el entrenamiento y otra para el test de los clasificadores. Para obtener el conjunto de entrenamiento se han escogido el 70 % de las muestras totales, mientras que para el test se han utilizado el 30 % restante. En la Tabla 5.1 se puede ver la distribución final de cada uno de los conjuntos.

5.1. Reducción de dimensionalidad

Los clasificadores se ven afectados cuando el número de dimensiones es muy grande. Por ello se realiza una reducción de dimensionalidad donde, además de reducir el tiempo de cómputo, se llega también a mejorar el reconocimiento, ya que se extraen aquellas componentes que presentan correlaciones más fuertes.

Al haber realizado el normalizado de la altura de las imágenes, se ha aumentado el número de dimensiones de los vectores de características hasta 14

píxeles por columna, añadiendo a estos vectores una gran cantidad de información que realmente no tiene mucha utilidad para el reconocimiento, pues tan solo se corresponden con líneas blancas. Para recuperar la información más relevante se ha aplicado una de las técnicas más comunes de reducción de dimensionalidad, conocida como PCA (por sus siglas en inglés, *Principal Component Analysis*) (Apéndice A.3).

Para comprobar la efectividad de PCA se han realizado múltiples entrenamientos para cada uno de los clasificadores, alterando en cada uno de los entrenamientos el número de dimensiones. Los tamaños de dimensiones utilizados han variado desde 10 hasta 150 realizando saltos de 10 unidades, utilizando un total de 15 dimensiones distintas.

5.2. *k* vecinos más cercanos

La implementación de este clasificador utilizando *Octave/Matlab* es realmente muy sencilla. Como ya se ha comentado, los paquetes de funciones estadísticas ofrecen una serie de métodos que facilitan mucho el trabajo. Como ejemplo de su simplicidad, en el Código 5.1 se muestra el código utilizado inicialmente para la clasificación mediante k vecinos más cercanos.

```
1  function [est] = knn (tr, trl, te, k)
2
3    [n, d] = size(tr);
4    [nt, dt] = size(te);
5
6    % Get minimun distances
7    D = pdist2(te, tr, 'euclidean');
8    [D, idx] = sort(D, 2, 'ascend');
9
10   % K-neares neighbors
11   D = D(:, 1:k);
12   idx = idx(:, 1:k);
13
14   % Majority vote
15   est = mode(trl(idx), 2);
16
17 end
```

Código 5.1: Clasificador k-NN en *Octave/Matlab*.

Para realizar la clasificación mediante k-NN se han lanzado varios experimentos variando en cada uno de ellos el número de dimensiones, y variando

el valor de k. Los valores de k utilizados se corresponden con los números impares comprendidos entre 1 y 20, obteniendo así 10 valores distintos.

Debido a que la ejecución se ha automatizado a través de un bucle, el Código 5.1 finalmente se ha modificado para reducir el tiempo de ejecución. En lugar de calcular la distancia para cada uno de los tamaños de k, éste cálculo se ha realizado una única vez y después se han escogido los k primeros vecinos en cada una de las iteraciones.

En total, para cada tamaño de ventana, se han obtenido 150 valores de error, producidos por combinar cada uno de los valores de las dimensiones (15 valores distintos a partir de PCA) con los 10 valores distintos utilizados de k.

5.3. Máquinas de vector soporte

Otro de los clasificadores utilizado para evaluar el reconocimiento de puntos han sido las máquinas de vector soporte. La clasificación de las muestras utilizando SVM se ha realizado utilizando una librería llamada *LibSVM*[4]. Esta librearía está disponible para multitud de lenguajes, entre los que se incluyen *Java*, *Python*, *C#*, *Octave/Matlab* o *Labview*.

Con este clasificador se realizaron varios experimentos iniciales variando la dimension del vector de características mediante PCA y variando las variables de las funciones utilizadas en el *Kernel*. Se probaron las funciones *Kernel* lineal, radial y sigmoid, pero finalmente solo se ha optado por utilizar la función radial, mostrada en la Tabla 3.1, dado que daba mejores resultados y realizaba el entrenamiento en menor tiempo.

Los valores de γ utilizados en la función radial se corresponden con $(n \cdot 10)^{-1}$, donde $n = \{1, 2, .., 10\}$. Al igual que en k-NN, se han obtenido un total de 150 valores de error para cada uno de los experimentos realizados con tamaños de ventana diferentes.

Capítulo 6

Evaluación

Es imprescindible realizar una evaluación de los clasificadores que permita comprobar la calidad de los mismos y, de este modo, poder comprobar cuál de los clasificadores funciona mejor y si realmente merece la pena utilizar alguno de ellos. Para poder evaluar el clasificador es necesario disponer de un corpus con el que realizar la evaluación. Cuando se realizó el entrenamiento en el Capítulo 5, el 30 % de las muestras quedó reservada para este fin.

La evaluación se lleva a cabo clasificando un conjunto de muestras ya etiquetadas y comprobando si el clasificador devuelve la etiqueta que corresponde. En este capítulo se evalúa cada clasificador con las muestras de test de cada uno de los tamaños de ventana, para comprobar cómo de bien se clasifican las muestras y cuál es el tamaño de ventana que mejor funciona para esta tarea, además de comprobar qué parámetros funcionan mejor con cada clasificador.

En cada una de las ejecuciones de evaluación que se han lanzado se han guardado los siguientes parámetros: el error obtenido; el número de dimensiones de PCA utilizado; el valor de γ o k, según corresponda para cada clasificador; el número de falsos positivos (FP); el número de falsos negativos (FN); y por último, el tiempo total requerido para realizar la clasificación. Indicar que en el tiempo guardado no se incluye el tiempo requerido para realizar el entrenamiento del clasificador, tan solo se ha guardado el tiempo utilizado en la clasificación.

A pesar de que el error es un parámetro más que suficiente para poder realizar la comparativa de los clasificadores, se ha decidido incluir los falsos positivos y los falsos negativos para poder realizar un análisis más exhaustivo de la clasificación. Los falsos positivos se corresponden con el número de

muestras que el clasificador ha etiquetado como *punto* cuando realmente no lo eran. Por otra parte, los falsos negativos son todas aquellas muestras que se etiquetan como *no punto* pero que en realidad sí lo eran y debían de haberse etiquetado como *punto*.

A partir de los falsos negativos y falsos positivos se pueden realizar el cálculo de la precisión, la sensibilidad o exhaustividad (en inglés, *recall*) y el valor-F (en inglés, *F-measures* o *F-score*). Estos valores son utilizados para comparar la relevancia de los distintos clasificadores utilizando distintos parámetros.

Mediante la precisión se comprueba la capacidad del clasificador de encontrar fragmentos de puntos, etiquetados como tal, sobre el conjunto total de puntos devueltos; por otra parte, la precisión permite comprobar la cantidad de puntos que el clasificador ha encontrado con respecto a la cantidad de puntos totales que habían disponibles en el corpus. El valor-F se calcula a través de la media armónica de la precisión y la sensibilidad, dándole a los dos valores la misma importancia.

Cabría recordar qué cuando se realizó la extracción del corpus, se obtuvieron un 20 % de muestras de *puntos* con respecto al 80 % de muestras de *no puntos*. Hay que tener en cuenta dichos valores, dado que un clasificador que siempre reconociera los fragmentos como *no punto* obtendría, utilizando este corpus, un error de clasificación del 20 %. Este error podría llegar a ser bastante menor si se tiene en cuenta que, en un texto normal, la relación de puntos que hay con respecto al resto de palabras es mucho menor al 20 %. Por ello aunque un sistema obtenga un error moderadamente bajo y parezca que clasifica bien, en realidad puede tratarse de un nefasto clasificador.

Se han realizado un total de 4500 ejecuciones distintas con el corpus de evaluación, variando en cada una de ellas el tamaño de la ventana utilizada para extraer los fragmentos, la dimensión del vector de características mediante PCA, y variando los valores de k y γ según el tipo de clasificador.

Pasemos a continuación a examinar los resultados obtenidos con cada uno de los clasificadores. Al final del capítulo se realizará, además, una comparativa donde se puedan ver las diferencias obtenidas en los resultados.

6.1. k vecinos más cercanos

El mejor resultado que se ha obtenido utilizando este clasificador es un error del 8,3 % utilizando una ventana de 25 píxeles de ancho, con una

reducción de la dimension mediante PCA de 130 dimensiones y un valor de $k = 1$. Comentar que, como más adelante podrá comprobarse, los resultados obtenidos con este clasificador son ligeramente superiores a los obtenidos con el clasificador SVM.

En la Figura 6.1 se puede ver cómo afecta el número de dimensiones obtenidos a partir de PCA en la clasificación. En esta gráfica se aprecian los mejores errores obtenidos con algunos tamaños de ventana y valor de k para cada una de las dimensiones PCA. Se puede ver claramente como a medida que se aumenta el número de dimensiones se mejora la clasificación, pero dado un número de dimensiones determinado el error no mejora y se mantiene prácticamente constante. También se puede apreciar que cuanto mayor contexto se incluye en la imagen, más tarda en converger el error devuelto por el clasificador.

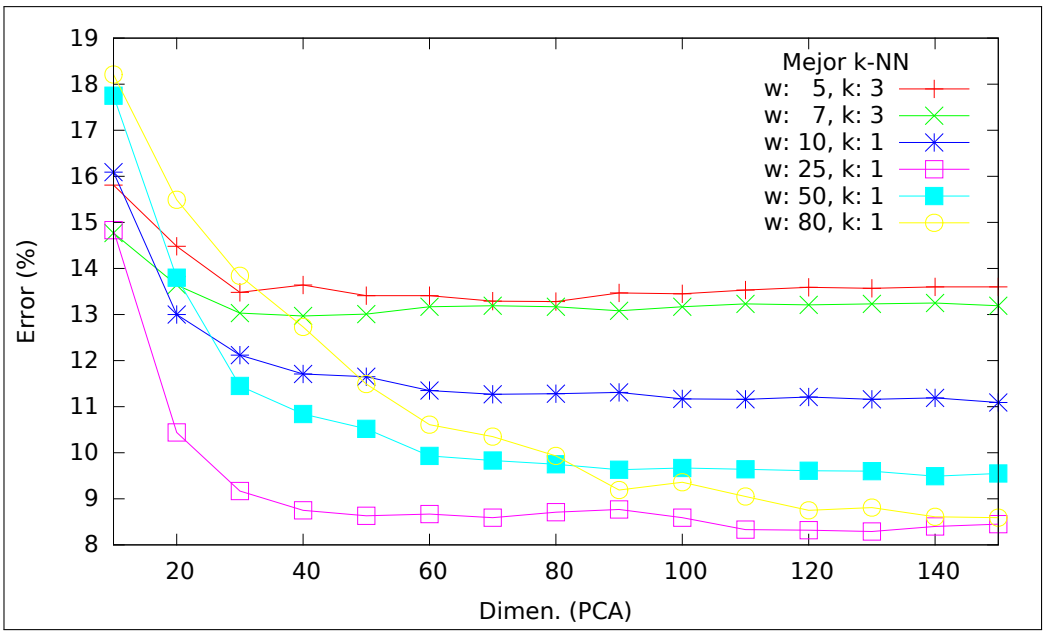


Figura 6.1: Algunos errores de clasificación en k-NN dependiendo de la dimensión de PCA para los mejores valores de k y tamaño de ventana.

Por otra parte, en la Figura 6.2 se muestra los mejores errores obtenidos con algunos de los tamaños de ventana y dimensión PCA para todos los valores de k. En este caso nos damos cuenta de que cuanto mayor es el tamaño de ventana la clasificación mejora al utilizar el menor número de vecinos. Esto puede ser debido al hecho de que cuanto mayor es la ventana ésta contiene más información que no se corresponde con el punto, si no con el contexto en el que éste se encuentra, y es más difícil encontrar varios puntos

que contengan un contexto similar si la ventana es muy grande.

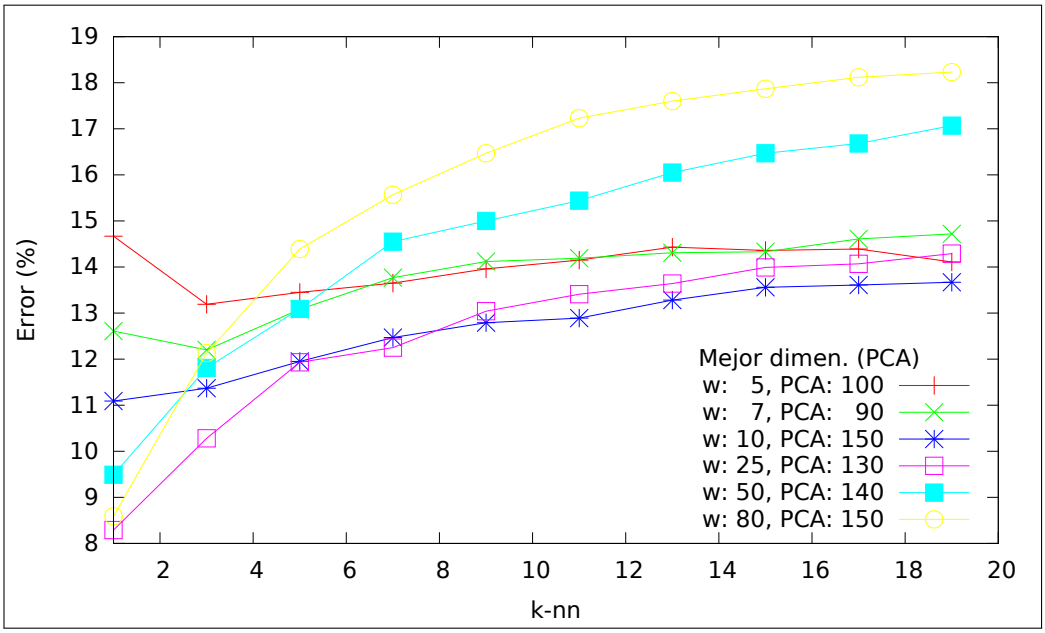


Figura 6.2: Algunos errores de clasificación en k-NN dependiendo de k para los mejores valores de dimensión de PCA y tamaño de ventana.

La Tabla 6.1 muestra los mejores resultados obtenidos para cada tamaño de ventana. Se puede ver como cuando el clasificador devuelve el error mínimo para cada tamaño de ventana, los valores de precisión, sensibilidad y valor-F son bastante similares, lo que nos permite comprobar que los parámetros utilizados para llegar a esos errores pueden ser unos buenos candidatos para utilizarlos en un futuro, dado que realizan una clasificación bastante equilibrada.

Dado que el objetivo final de estos clasificadores es el de ayudar a un sistema a generar frases que estén, gramaticalmente, lo mejor formadas, quizá sería lógico intentar obtener un sistema con la máxima precisión posible, es decir, que de entre los puntos que encuentre no haya ningún falso positivo. De este modo se evitaría dividir líneas cuya estructura gramatical era correcta y que, en el caso de dividirlas, perderían parte de su sentido o dejaría de seguir la gramática de la lengua del texto.

La Tabla 6.2 nos muestra cómo cuando la precisión es mayor la sensibilidad disminuye considerablemente, por lo que hay una gran cantidad de puntos que no se están reconociendo como tal, y por consiguiente el error también aumenta. Pero de este modo nos aseguramos de que en el mejor de los casos el 88.6 % de los fragmentos etiquetados como puntos se corresponden

realmente con un punto.

En el caso de la sensibilidad, en la Tabla 6.3 se puede apreciar como cuando se obtienen los mejores resultados, la diferencia con la precisión no es tan grande y se obtienen unos valores bastante similares, aunque cabe destacar que cuanto mayor es la sensibilidad, la diferencia con respecto a la precisión aumenta. Esto nos indica que el clasificador k-NN tiende a clasificar las muestras más como *punto* que como *no punto*, dado que se reduce más el número de falsos negativos que de falsos positivos.

Los valores obtenidos en la Tabla 6.4 son prácticamente los mismos que los obtenidos en la Tabla 6.1. El error y los valores-F están inversamente relacionados, ya que al ser el valor-F la relación entre la precisión y la sensibilidad, cuanto mayores sean éstas mejor funcionará el clasificador.

Por último cabría destacar que, dado que el número de dimensiones de PCA son bastante similares cuando se dan los mejores resultados, no se encuentran diferencias significativas de tiempo que puedan llegar a descartar algunas de las configuraciones, a no ser que en un futuro sistema de reconocimiento por frases el tiempo de ejecución resulte una variable crítica.

6.2. Máquinas de vector soporte

En el caso de las máquinas de vector soporte el mejor resultado se obtiene utilizando una ventana de 20 píxeles de ancho, reduciendo el vector de características mediante PCA a 90 dimensiones y utilizando un valor γ en la función radial de 20^{-1}. Con estos parámetros el mejor error alcanzado es de 9,2 %.

Al igual que se ha comprobado en los experimentos utilizados en k-NN, en la Figura 6.3 se aprecia cómo afecta la reducción de la dimension utilizando PCA, donde el error deja de mejorar a partir de un determinado número de dimensiones.

Window	PCA	k	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	100	3	13.2	69.0	61.9	65.3	31.5
7	90	3	12.2	71.8	64.3	67.8	30.0
10	150	1	11.1	70.1	77.7	73.7	40.2
12	90	1	9.4	73.6	82.1	77.7	28.8
15	130	1	9.2	74.5	82.0	78.0	35.7
20	140	1	8.6	77.3	80.7	79.0	38.9
25	130	1	**8.3**	78.3	80.9	79.6	36.7
30	110	1	8.4	79.0	78.8	78.9	32.6
40	90	1	9.3	77.7	74.7	76.2	30.6
50	140	1	9.5	78.5	72.3	75.3	39.3
60	130	1	9.6	78.0	72.8	75.3	36.9
70	110	1	9.2	77.1	77.3	77.2	33.7
80	150	1	8.6	80.3	75.6	77.9	42.3
90	140	1	9.9	75.0	76.0	75.5	42.9
100	140	1	10.2	74.0	75.4	74.7	40.1

Tabla 6.1: Error mínimo obtenido en k-NN para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	PCA	k	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	50	19	14.0	74.2	45.8	56.6	20.8
7	20	13	14.1	74.1	45.1	56.1	13.2
10	30	13	13.1	78.0	48.2	59.6	16.1
12	60	5	10.9	79.3	61.7	69.4	22.5
15	70	17	13.2	78.9	46.5	58.5	25.4
20	150	15	13.3	81.8	42.9	56.3	41.3
25	70	17	13.6	82.7	40.2	54.1	24.0
30	90	15	14.2	85.1	34.9	49.5	28.4
40	140	9	14.2	86.1	34.3	49.0	38.8
50	150	5	13.2	84.4	41.9	56.0	41.7
60	80	13	16.0	85.2	24.1	37.6	27.8
70	150	19	17.8	86.1	13.3	23.0	42.0
80	110	15	17.1	**88.6**	16.6	28.0	33.6
90	90	19	17.2	84.2	17.5	28.9	32.2
100	130	7	15.9	78.5	28.3	41.6	37.9

Tabla 6.2: Precisión máxima obtenida en k-NN para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	PCA	k	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	80	1	14.5	62.9	66.7	64.7	27.5
7	90	1	12.6	67.2	72.2	69.6	30.0
10	130	1	11.2	69.7	78.1	73.7	37.1
12	150	1	9.5	73.3	**82.5**	77.6	40.2
15	140	1	9.2	74.3	82.2	78.0	38.1
20	50	1	8.8	76.5	80.7	78.5	20.8
25	110	1	8.3	78.1	81.1	79.6	32.4
30	40	1	8.6	77.5	80.3	78.9	18.7
40	40	1	9.6	75.5	76.7	76.1	18.5
50	70	1	9.8	76.2	73.9	75.1	25.2
60	50	1	11.2	71.1	74.4	72.7	21.1
70	100	1	9.3	76.3	77.5	76.9	31.5
80	60	1	10.6	71.8	77.4	74.5	23.2
90	70	1	11.6	68.6	77.3	72.7	27.8
100	90	1	11.6	69.1	76.4	72.6	29.5

Tabla 6.3: Sensibilidad máxima obtenida en k-NN para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	PCA	k	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	90	3	13.2	69.0	61.9	65.3	29.5
7	90	1	12.6	67.2	72.2	69.6	30.0
10	150	1	11.1	70.1	77.7	73.7	40.2
12	90	1	9.4	73.6	82.1	77.7	28.8
15	140	1	9.2	74.3	82.2	78.0	38.1
20	140	1	8.6	77.3	80.7	79.0	38.9
25	130	1	8.3	78.3	80.9	**79.6**	36.7
30	50	1	8.5	78.1	79.9	79.0	20.3
40	90	1	9.3	77.7	74.7	76.2	30.6
50	140	1	9.5	78.5	72.3	75.3	39.3
60	130	1	9.6	78.0	72.8	75.3	36.9
70	110	1	9.2	77.1	77.3	77.2	33.7
80	150	1	8.6	80.3	75.6	77.9	42.3
90	140	1	9.9	75.0	76.0	75.5	42.9
100	140	1	10.2	74.0	75.4	74.7	40.1

Tabla 6.4: Valor-F máximo obtenido en k-NN para cada tamaño de ventana. En negrita se resalta el mejor resultado.

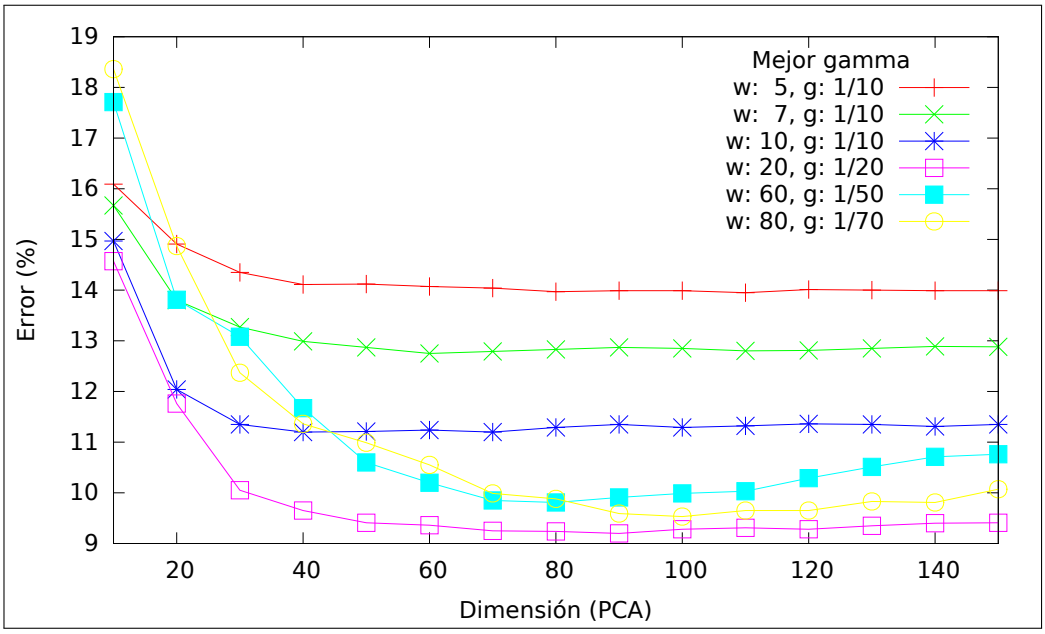


Figura 6.3: Algunos errores de clasificación en SVM dependiendo de la dimensión de PCA para los mejores valores de γ(g) y tamaño de ventana.

Por otro lado, la Figura 6.4 muestra los mejores errores obtenidos para algunos tamaños de ventana y dimensión PCA para todos los valores de γ utilizados. Se aprecia como cuanto mayor es el ancho de la ventana, el mejor error obtenido se consigue con un valor de γ más bajo. Esto es debido a la propia definición de γ, ya que al ser inverso a la varianza, cuanto mayor es el tamaño de ventana, se produce una mayor dispersión y varianza de los datos; por lo tanto menor es el tamaño de γ que se ajusta a los datos.

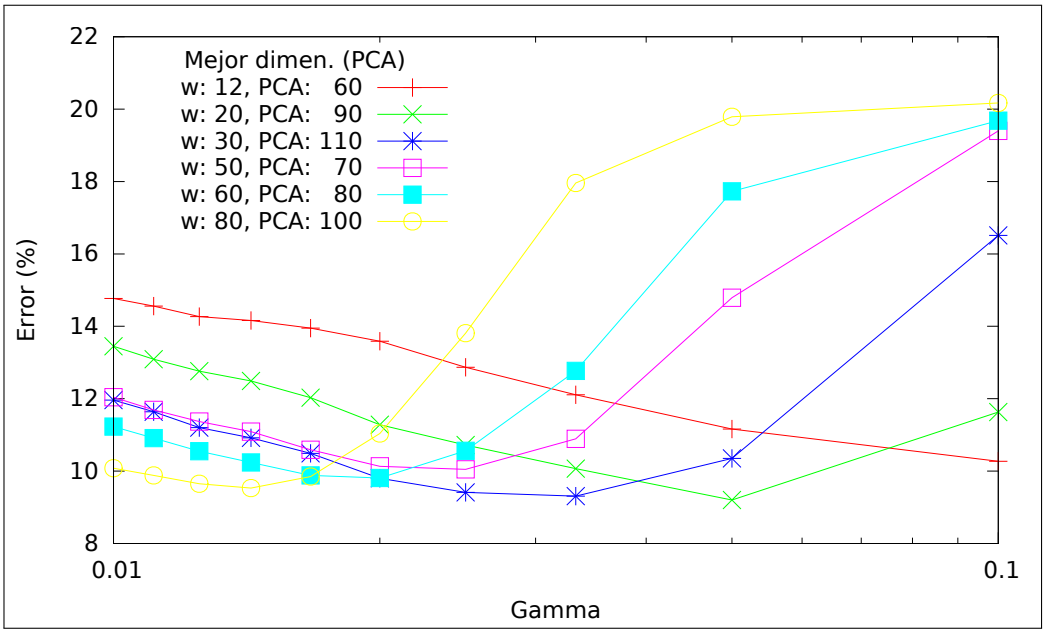


Figura 6.4: Algunos errores de clasificación en SVM dependiendo de γ para los mejores valores de dimensión de PCA y tamaño de ventana.

En la Tabla 6.5 se muestra un resumen de los resultados en los que el error obtenido ha sido mínimo. A diferencia de lo que ocurría en el clasificador k-NN, en este clasificador sí se observan disparidades entre los valores de precisión y similitud. Este clasificador comete menos errores a la hora de elegir un fragmento como punto, y aunque el error obtenido sea mayor que en el clasificador k-NN, quizás debido a esto merecería más la pena utilizar este clasificador.

Como se ha comentado antes, pensando en el trabajo futuro al que está destinado este clasificador, puede ser conveniente buscar unos parámetros que devuelvan la mejor precisión posible. En este caso el clasificador SVM es capaz de obtener una precisión realmente alta, como se puede ver en la Tabla 6.6. Con la mayor precisión obtenida, el clasificador tan solo se equivoca en el 2,2 % de los fragmentos etiquetados como punto, pero si se analizan los demás parámetros podemos comprobar que lo realiza a causa de etiquetar casi siempre las muestras como *no punto*; por ello, el error obtenido es prácticamente el porcentaje de muestras de *punto* utilizados en el corpus de entrenamiento.

Uno de los resultados más interesantes se obtiene cuando se analiza la Tabla 6.7, donde a pesar de no obtener el error mínimo, los parámetros utilizados para obtener la máxima sensibilidad parecen unos buenos candidatos a

utilizar en el sistema de reconocimiento final. Si se analiza detenidamente, la precisión obtenida es bastante alta, y aunque el resto de valores sean bastante similares a los adquiridos con los parámetros que hacen el error mínimo, la principal ventaja de esta configuración reside en el tiempo de ejecución, donde éste es menor. Esto es debido a que utiliza un menor número de dimensiones y por tanto realiza menos cálculos para realizar la clasificación.

Por último, al igual que ocurría con el clasificador k-NN, los resultados obtenidos con el valor-F máximo son prácticamente los mismos a los obtenidos con el error mínimo, tal y como puede apreciarse al comparar la Tabla 6.8 con la Tabla 6.5.

6.3. Comparativa de clasificadores

Como se puede comprobar en los resultados de error obtenidos, k-NN se comporta mejor que SVM para esta tarea, dando el mejor resultado de k-NN un error del 8,3 %, mientras que el mejor resultado de SVM obtiene un error del 9,2 %. En la Tabla 6.9 se pueden ver los mejores resultados que se han obtenido para cada una de las ventanas, mostrando además los valores de las dimensiones PCA y los valores de k y γ según corresponda, con los que se han obtenido dichos resultados.

Por otra parte, tampoco se puede elegir el clasificador por su error mínimo ya que, como se ha comentado, puede que para el sistema de reconocimiento orientado por frases lo más conveniente sea elegir el clasificador por la precisión y la sensibilidad que éste obtiene. En este caso se podría optar por el clasificador SVM ya que éste comete menos errores al etiquetar un *no punto* como *punto* y la diferencia de error no es relativamente muy superior al otro clasificador.

Window	PCA	γ^{-1}	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	110	10	13.9	83.8	37.5	51.8	18.4
7	60	10	12.8	84.0	44.8	58.4	12.2
10	40	10	11.2	85.2	53.3	65.5	9.8
12	60	10	10.3	86.8	57.4	69.1	14.9
15	100	20	10.5	85.2	57.3	68.6	19.9
20	90	20	**9.2**	89.5	61.2	72.7	20.4
25	60	20	9.4	88.4	61.2	72.3	15.9
30	110	30	9.3	88.8	61.2	72.5	25.4
40	90	40	9.7	87.9	60.0	71.3	22.6
50	70	40	10.1	88.8	56.9	69.4	18.8
60	80	50	9.8	91.5	56.1	69.6	22.4
70	130	80	9.6	90.1	58.4	70.9	31.1
80	100	70	9.5	92.2	57.1	70.6	27.0
90	130	100	9.9	91.1	55.7	69.1	32.3
100	70	70	10.7	89.3	53.0	66.5	19.6

Tabla 6.5: Error mínimo obtenido en SVM para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	PCA	γ^{-1}	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	60	40	16.5	84.1	21.5	34.2	10.8
7	70	10	12.8	84.1	44.5	58.2	12.8
10	140	10	11.3	85.7	52.2	64.9	30.1
12	150	10	10.5	87.4	55.4	67.8	32.7
15	150	10	11.2	88.5	50.6	64.4	40.0
20	130	10	12.1	93.3	42.5	58.4	41.0
25	120	10	14.0	92.4	32.5	48.1	42.5
30	90	10	16.0	96.0	20.9	34.4	33.4
40	90	10	18.5	95.9	7.9	14.5	36.7
50	150	30	13.5	93.9	34.8	50.8	49.3
60	60	20	16.0	96.3	20.7	34.1	26.6
70	100	30	16.3	97.3	19.2	32.1	38.6
80	120	40	15.0	98.4	25.3	40.3	43.9
90	150	50	14.9	95.7	26.7	41.7	62.6
100	120	30	19.4	**97.8**	3.0	5.8	46.3

Tabla 6.6: Precisión máxima obtenida en SVM para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	PCA	γ^{-1}	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	90	10	14.0	83.5	37.5	51.7	16.1
7	60	10	12.8	84.0	44.8	58.4	12.2
10	40	10	11.2	85.2	53.3	65.5	9.8
12	60	10	10.3	86.8	57.4	69.1	14.9
15	70	20	10.6	84.8	57.5	68.5	14.5
20	70	20	9.2	89.0	61.3	72.6	15.9
25	40	20	9.6	86.7	61.5	72.0	10.5
30	60	30	9.4	87.2	**62.0**	72.5	13.6
40	70	40	9.7	86.9	60.6	71.4	16.4
50	70	50	10.1	87.0	58.0	69.6	17.3
60	80	60	9.9	90.1	56.9	69.7	21.0
70	130	80	9.6	90.1	58.4	70.9	31.1
80	120	90	9.7	90.2	57.9	70.5	28.7
90	100	90	10.0	90.1	56.4	69.4	24.9
100	80	90	10.9	86.7	53.5	66.2	19.9

Tabla 6.7: Sensibilidad máxima obtenida en SVM para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	PCA	γ^{-1}	Error (%)	Precisión (%)	Recall (%)	F-score (%)	Tiempo (seg.)
5	110	10	13.9	83.8	37.5	51.8	18.4
7	60	10	12.8	84.0	44.8	58.4	12.2
10	40	10	11.2	85.2	53.3	65.5	9.8
12	60	10	10.3	86.8	57.4	69.1	14.9
15	100	20	10.5	85.2	57.3	68.6	19.9
20	90	20	9.2	89.5	61.2	**72.7**	20.4
25	60	20	9.4	88.4	61.2	72.3	15.9
30	90	40	9.4	87.6	61.8	72.5	19.4
40	70	40	9.7	86.9	60.6	71.4	16.4
50	70	50	10.1	87.0	58.0	69.6	17.3
60	80	60	9.9	90.1	56.9	69.7	21.0
70	130	80	9.6	90.1	58.4	70.9	31.1
80	120	80	9.6	91.5	57.5	70.7	30.5
90	100	90	10.0	90.1	56.4	69.4	24.9
100	70	80	10.8	88.0	53.5	66.6	18.0

Tabla 6.8: Valor-F máximo obtenido en SVM para cada tamaño de ventana. En negrita se resalta el mejor resultado.

Window	k-NN			SVM		
	PCA	k	Error (%)	PCA	γ^{-1}	Error (%)
5	100	3	13.2	110	10	14.0
7	90	3	12.2	60	10	12.8
10	150	1	11.1	40	10	11.2
12	90	1	9.5	60	10	10.3
15	130	1	9.2	100	20	10.5
20	140	1	8.6	90	20	**9.2**
25	130	1	**8.3**	60	20	9.4
30	110	1	8.4	110	30	9.3
40	90	1	9.4	90	40	9.7
50	140	1	9.5	70	40	10.1
60	130	1	9.6	80	50	9.8
70	110	1	9.2	130	80	9.6
80	150	1	8.6	100	70	9.5
90	140	1	9.9	130	100	10.0
100	140	1	10.2	70	70	10.7

Tabla 6.9: Mejores resultados obtenidos para cada tamaño de ventana en ambos clasificadores. En negrita se resalta el mejor resultado de cada clasificador.

Capítulo 7

Conclusión

Este trabajo presenta un primer paso para llegar en el futuro a un sistema de reconocimiento de texto basado en frases completas. Tras comprobar los resultados obtenidos, queda claro que los clasificadores k-NN y SVM resultan apropiados para realizar la detección de puntos, y parecen buenos candidatos a utilizar en el sistema encargado de detectar los límites de las frases en las imágenes de líneas.

Por otra parte, se ha podido comprobar como la cantidad de información utilizada para realizar la clasificación influye en este tipo de clasificadores, obteniendo los mejores resultados cuando la ventana utilizada para la extracción de fragmentos tiene un ancho de entre 20 y 25 píxeles. Además, también se ha visto cómo afecta la reducción de la dimensión de los vectores de características en ambos clasificadores. En las pruebas realizadas los mejores resultados se han obtenido utilizando 90 y 130 dimensiones.

También se ha podido experimentar cómo varían la sensibilidad y la precisión en estos clasificadores, y cómo el error mínimo puede no ser siempre la mejor variable a tener el cuenta a la hora de escoger un sistema de reconocimiento.

A pesar de haber obtenido unos buenos resultados, creo que pueden ser fácilmente mejorables si se mejora el proceso de extracción de fragmentos a la hora de obtener el corpus. En primer lugar, las posiciones de los puntos extraídos mediante el reconocimiento forzado no son del todo correctas, y a pesar de que se haya supervisado manualmente, muchas de las posiciones no son del todo precisas. Además, existen líneas donde la transcripción asegura que en la imagen hay un punto, cuando realmente al visualizar la imagen éste no está presente. Lo mismo ocurre en el caso contrario, cuando al inspeccionar

ciertas imágenes nos encontramos con puntos que en la transcripción no se habían reflejado.

En segundo lugar, se puede mejorar el reconocimiento cambiando el modo de etiquetado de los fragmentos de *punto*. En el proceso realizado en este trabajo, un fragmento *punto* es etiquetado como tal cuando cualquier columna que lo compone entra dentro de las fronteras de un punto. Las posiciones extraídas en el reconocimiento forzado contienen un cierto umbral de error, por lo que las posiciones de un punto no se ciñen únicamente al punto, si no que contienen parte del contexto en el que éste se encuentre. Debido a esto, en el proceso de etiquetado de los puntos hay casos en los que los fragmentos son etiquetados como *punto* cuando en realidad no lo son. Ésto podría solucionarse etiquetando un punto como tal cuando un cierto número de columnas o un porcentaje de las mismas se encuentre dentro de las fronteras de un punto.

El incluir estas dos mejoras en el sistema podría llegar a mejorar los resultados obtenidos, pero para poder comprobarlo es necesario realizar de nuevo una evaluación exhaustiva probando diferentes combinaciones, aumentando así el número de configuraciones posibles y dificultando más el análisis de los resultados.

7.1. Trabajo futuro

Este trabajo final de máster es parte de un proyecto cuyo objetivo final es realizar un sistema de reconocimiento por frases. Dado que este trabajo solo presenta una parte de dicho proyecto, todavía queda mucho trabajo por completar.

Además, dentro de lo que sería la parte de la detección de puntos, aún puede ser mejorada añadiendo la diferenciación de qué tipo de punto se trata. Por ejemplo, cuando se trata de un punto debido a la abreviación de un nombre o el tratamiento de una persona no se debería dividir la línea dado que se trata de la misma sentencia. Así mismo, también habría que mejorar el número de signos de puntuación que es capaz de reconocer el clasificador para poder incluir los guiones utilizados para dividir una palabra que no cabe en una línea; de este modo, se podrían fusionar líneas eliminando dicho guión y facilitando así la tarea de reconocimiento.

Dado que se han comprobado diferencias en la precisión y la sensibilidad de los clasificadores, sería interesante en un futuro realizar la evaluación del sistema de reconocimiento por frases utilizando distintas configuraciones del

sistema de reconocimiento de signos de puntuación, donde se premie más que se reconozcan correctamente los puntos sin confundirlos con otros caracteres a realizar el mejor reconocimiento posible.

Como trabajo futuro quedaría, además, integrar este trabajo dentro de un sistema que sea capar de dividir y fusionar las imágenes de las líneas consiguiendo así formar una única imagen, la cual contenga la sentencia completa que posteriormente será utilizada en el sistema de reconocimiento de texto manuscrito.

Apéndice A

Extracción del corpus

A.1. iATROS

iATROS[7] es un sistema de reconocimiento de habla e imágenes manuscritas, desarrollado por el centro de investigación *Pattern Recognition and Human Language Technology* (PRHLT) de la Universitat Politècnica de València. iATROS proviene de la mejora de un reconocedor de habla anterior (ATROS), el cual ha sido adaptado para poder reconocer tanto el habla como texto manuscrito.

Este programa permite extraer los vectores de características, generar los modelos de lenguaje y realizar las transcripciones, tanto para texto manuscrito como para grabaciones de audio, lo cual lo convierte en una herramienta muy versátil.

A.2. Gramáticas FSM

Las gramáticas FSM (por sus siglas del ingles, *Finite State Machine*) al igual que los n-gramas se utilizan para definir modelos de lenguaje. Están formadas por autómatas de estados finitos en los que cada transición de pasar de un estado a otro genera una palabra. Éstas transiciones tienen además asociada la probabilidad de que cada palabra suceda.

Para poder realizar el reconocimiento forzado es necesario crear gramáticas que solo sean capaz de reconocer una única frase: la frase de la imagen a transcribir. Por ello las gramáticas generadas para este trabajo son muy sencillas y tan solo tienen una única transición entre cada uno de sus esta-

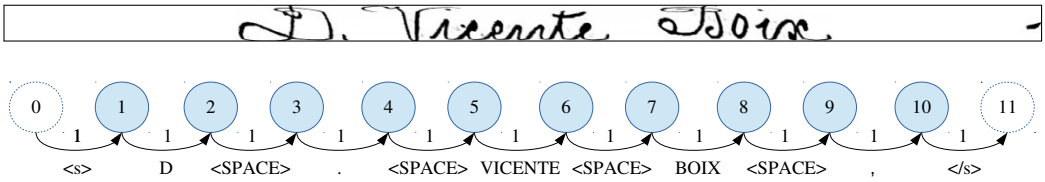


Figura A.1: Gráfico de la gramática FSM de la frase "D. Vicente Boix,".

dos. No obstante, las gramáticas FSM no solo sirven para generar gramáticas sencillas, sino que permiten llegar a definir gramáticas realmente complejas

En el Código A.1 se puede ver un ejemplo de una de las gramáticas FSM utilizadas en este trabajo, en la que se muestra la gramática generada para la frase "D. Vicente Boix,". Como puede verse, ha sido necesaria la adición de estados para representar el inicio y final de la frase, además de representar los espacios entre palabras como el símbolo especial "<SPACE>". En la Figura A.1 puede verse el autómata de estados finitos del ejemplo anterior de forma gráfica.

```
state 0 c=1 i=1
0 1 "<s>" p=1
state 1 c=1
1 2 "D" p=1
state 2 c=1
2 3 "<SPACE>" p=1
state 3 c=1
3 4 "." p=1
state 4 c=1
4 5 "<SPACE>" p=1
state 5 c=1
5 6 "VICENTE" p=1
state 6 c=1
6 7 "<SPACE>" p=1
state 7 c=1
7 8 "BOIX" p=1
state 8 c=1
8 9 "<SPACE>" p=1
state 9 c=1
9 10 "," p=1
state 10 c=1
10 11 "</s>" p=1
state 11 c=1 f=1
```

Código A.1: Gramática FSM de la frase "D. Vicente Boix,".

A.3. Principal Component Analysis

El análisis de las componentes principales o PCA (por sus siglas del inglés, *Principal Component Analisis*) permite reducir el número de dimensiones de un vector mediante la proyección de éste a un sistema de representación inferior. Por lo tanto, su objetivo es encontrar una matriz de proyección W que sea capaz de minimizar el error que se produce al realizar dicha proyección.

El error de proyección de un vector x en un espacio original se mide realizando la proyección a un espacio más reducido, $\hat{x}$, para después volver a proyectarlo al espacio original y poder comprobar así su diferencia. El error de proyección de un conjunto de n muestras $x_1, x_2, \ldots, x_n$ proyectadas en k dimensiones, se define como el error cuadrático:

$$error_k = \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{A.1}$$

El cálculo de la matriz de proyección W para reducir el número de dimensiones mediante PCA a k dimensiones en un conjunto de n vectores $x = \{x_1, x_2, ..., x_n\}$, se calcula mediante el siguiente algoritmo:

1. Calcular la media de los datos: $\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$
2. Restar a todos los datos la media: $x_i \leftarrow x_i - \overline{x}$
3. Calcular la matriz de covarianzas $\sum_x = \frac{1}{n} \sum_{i=1}^{n} x_i (x_i)^t$
4. Calcular los eigenvectores de la matriz de covarianzas $\sum_x$
5. Ordenar de mayor a menor los eigenvectores según sus eigenvalores asociados.
6. Definir W como una matriz formada por los k primeros eigenvectores

A.4. HTK Standard Lattice Format (SLF)

Los ficheros SLF (por sus siglas en inglés, *Standard Lattice Format*) de HTK permiten almacenar las hipótesis obtenidas de un sistema de reconocimiento utilizando el autómatas de estados finitos de la gramática de la frase reconocida. El fichero almacena, entre otras cosas, la primera posición o el instante en el que aparece cada nodo junto con las transiciones ordenadas que hay entre éstos. De este modo se pueden calcular en una imagen la posición

que ocupa cada una de las palabras reconocidas.

En el Código A.2 se muestra un ejemplo del fichero de lattice obtenido a partir del reconocimiento forzado utilizando iATROS. Dicho código se ha obtenido después de forzar al sistema a reconocer una imagen utilizando la gramatica FSM del Código A.1.

```
# Word graph in SLF format generated by iAtros
UTTERANCE=./lattices/01.nobackground.noskew_07.fea.lat.gz
lmscale=100.00
wdpenalty=-0.00
wdpenalty_output=-0.00
# Size line
N=13 L=12
# Node definitions
I=0 t=0
I=1 t=2
I=2 t=16
I=3 t=18
I=4 t=32
I=5 t=34
I=6 t=526
I=7 t=578
I=8 t=751
I=9 t=879
I=10 t=952
I=11 t=1193
I=12 t=1193
J=0 S=0 E=1 W=<s> a=275.209351 l=0.000000 O=
J=1 S=1 E=2 W=D a=554.706970 l=0.000000 O=
J=2 S=2 E=3 W=<SPACE> a=254.008575 l=0.000000 O=
J=3 S=3 E=4 W=. a=1382.731812 l=0.000000 O=
J=4 S=4 E=5 W=<SPACE> a=254.008575 l=0.000000 O=
J=5 S=5 E=6 W=VICENTE a=42100.906250 l=0.000000 O=
J=6 S=6 E=7 W=<SPACE> a=2055.193115 l=0.000000 O=
J=7 S=7 E=8 W=BOIX a=-1780.174072 l=0.000000 O=
J=8 S=8 E=9 W=<SPACE> a=1902.630859 l=0.000000 O=
J=9 S=9 E=10 W=, a=3995.103271 l=0.000000 O=
J=10 S=10 E=11 W=</s> a=32171.746094 l=0.000000 O=
J=11 S=11 E=12 W=!NULL a=0.000000 l=0.000000 O=!NULL c
   =53114912
```

Código A.2: Fichero SLF donde se reconoce la frase “D. Vicente Boix”

Índice de códigos

5.1. Clasificador k-NN en *Octave/Matlab*. 29
A.1. Gramática FSM de la frase "D. Vicente Boix,". 48
A.2. Fichero SLF donde se reconoce la frase "D. Vicente Boix" . . . 50

Índice de figuras

2.1. Tableta digitalizadora utilizada para el reconocimiento *online*. 9
2.2. Líneas del corpus "Cristo Salvador". 11
2.3. Líneas del corpus "Cristo Salvador" normalizadas. 12
2.4. Ejemplo de un Modelo Oculto de Markov discreto. 13

3.1. Ejemplo de un clasificador k-NN 17
3.2. Ejemplo de un clasificador SVM 18

4.1. Páginas extraidas del corpus "Cristo Salvador". 21
4.2. Ejemplo de una línea del corpus "Cristo Salvador". 22
4.3. Líneas con errores de transcripción. 24
4.4. Línea de altura máxima junto con una línea normalizada a la misma altura. 25

6.1. Algunos errores de clasificación en k-NN dependiendo de la dimensión de PCA para los mejores valores de k y tamaño de ventana. 33
6.2. Algunos errores de clasificación en k-NN dependiendo de k para los mejores valores de dimensión de PCA y tamaño de ventana. 34
6.3. Algunos errores de clasificación en SVM dependiendo de la dimensión de PCA para los mejores valores de γ(g) y tamaño de ventana. 38
6.4. Algunos errores de clasificación en SVM dependiendo de γ para los mejores valores de dimensión de PCA y tamaño de ventana. 39

A.1. Gráfico de la gramática FSM de la frase "D. Vicente Boix,". . 48

Índice de tablas

3.1. Funciones Kernel utilizadas en SVM. 19

4.1. Datos sobre el corpus “Cristo Salvador”. 22
4.2. Número de fragmentos obtenidos con distintos tamaños de ventana 26

5.1. Distribución de muestras de los conjuntos de entrenamiento y evaluación 28

6.1. Error mínimo obtenido en k-NN para cada tamaño de ventana. 36
6.2. Precisión máxima obtenida en k-NN para cada tamaño de ventana. 36
6.3. Sensibilidad máxima obtenida en k-NN para cada tamaño de ventana. 37
6.4. Valor-F máximo obtenido en k-NN para cada tamaño de ventana. 37
6.5. Error mínimo obtenido en SVM para cada tamaño de ventana. 41
6.6. Precisión máxima obtenida en SVM para cada tamaño de ventana. 41
6.7. Sensibilidad máxima obtenida en SVM para cada tamaño de ventana. 42
6.8. Valor-F máximo obtenido en SVM para cada tamaño de ventana. 42
6.9. Mejores resultados obtenidos para cada tamaño de ventana en ambos clasificadores. En negrita se resalta el mejor resultado de cada clasificador. 43

Bibliografía

[1] Biblioteca Valenciana Digital (BIVALDI). *Noticia histórica de las fiestas con que Valencia celebró el siglo sexto de la venida a esta capital de la milagrosa imagen del Salvador*. 2015. URL: http://bivaldi.gva.es/es/consulta/registro.cmd?id=281.

[2] Peter F Brown y col. "Class-based n-gram models of natural language". En: *Computational linguistics* 18.4 (1992), págs. 467-479.

[3] Horst Bunke, Samy Bengio y Alessandro Vinciarelli. "Offline recognition of unconstrained handwritten texts using HMMs and statistical language models". En: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.6 (2004), págs. 709-720.

[4] Chih-Chung Chang y Chih-Jen Lin. "LIBSVM: A library for support vector machines". En: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software disponible en http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1-27:27.

[5] Departament of Engineering, University of Cambridge. *HTK Speech recognition tookit*. 2016. URL: http://htk.eng.cam.ac.uk/docs/docs.shtml.

[6] Louisa Lam y Ching Y. Suen. "Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers". En: *Pattern Recognition* 21.1 (1988), págs. 19-31.

[7] Míriam Luján-Mares y col. "iATROS: A speech and handwritting recognition system." En: *V Jornadas en Tecnologías del Habla (VJTH'2008)*. 2008, págs. 75-78.

[8] U-V Marti y Horst Bunke. "Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system". En: *International journal of Pattern Recognition and Artificial intelligence* 15.01 (2001), págs. 65-90.

[9] Verónica Romero y col. “Computer assisted transcription for ancient text images”. En: *Image Analysis and Recognition*. Springer, 2007, págs. 1182-1193.

[10] BW Silveman y col. “An important contribution to nonparametric discriminant analysis and density estimation”. En: *International Statistical Review* 57.3 (1951), págs. 233-247.

[11] Alejandro Hector Toselli, Alfons Juan y Enrique Vidal. “Spontaneous Handwriting Recognition and Classification.” En: *ICPR (1)*. 2004, págs. 433-436.