VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNAM

Unidad Académica de los Ciclos Profesional y de Posgrado del Colegio de Ciencias y Humanidades

Un sistema de reconocimiento óptico de caracteres

TESIS

que para obtener el grado de

Maestro en Ciencias de la computación

presenta

Edgar Leonel Chávez González

México, D.F. Agosto de 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Un sistema de reconocimiento óptico de caracteres

1

Introducción

1.1 Antecedentes

Una de las tareas que realizamos los humanos con mayor frecuencia es la discriminación de objetos, constantemente, sin prestar atención, observamos, escuchamos y sentimos. A lo largo de siglos de evolución, nuestra raza ha aprendido a establecer con claridad la identidad de estas sensaciones. A través de los órganos sensores, podemos recrear el mundo exterior identificando componentes; más aún, podemos clasificar los objetos, los sonidos, las texturas, los aromas y los sabores de tal modo que encontramos simultáneamente un grupo al que pertenecen y las diferencias de esa sensación particular con los demás elementos del grupo. A la mente humana le toma muy poco tiempo este proceso, nos parece instantáneo y es una de las cosas que no merecen nuestra atención cotidiana.

Por otro lado, existe la intención de mecanizar este proceso de discriminación, y los motivos van desde el prurito meramente académico hasta la resolución de tareas fatigosas para un ser humano en aplicaciones del mundo real.

Las tareas de percepción pueden segmentarse en dos etapas globales, la obtención de una representación numérica de la escena (en un sentido amplio, no meramente visual) y el uso intensivo de relaciones numéricas entre representaciones para la clasificación. Las representaciones usualmente son arreglos numéricos crudos obtenidos de los sensores; no hay garantía, sin embargo, de que la representación directa sea la adecuada para un tratamiento de discriminación de eficiencia comparable a la humana, ya no digamos cuando se trata de obtener resultados seme-

jantes en cuanto a velocidad, sino cuando queremos comparar el número de errores cometidos.

La búsqueda de representaciones adecuadas de las escenas sensoriales ha sido objeto de numerosos estudios interdiciplinarios. Sin embargo, la representación está estrechamente ligada al paradigma de cálculo, el cual no ha sido descifrado hasta la fecha; no sabemos con precisión que sucede en el interior de las células neurales.

El proceso de discriminación, agrupación y clasificación de información forma parte de nuestros actos cotidianos; los actos puramente intelectuales pueden ser vistos como la búsqueda de agrupaciones, relaciones entre agrupaciones y búsqueda de patrones o invariantes entre objetos.

Cuando una sensación no ha sido registrada y es la primera vez que se presenta ante nosotros, somos capaces de abrir una página nueva en donde vamos acumulando información acerca de ella, pero esta sensación no se guarda sin referencias a sensaciones anteriores; al mismo tiempo que registramos la sensación estamos registrando sus relaciones con las sensaciones que tenemos almacenadas. Este proceso se sublima; el cerebro es capaz de clasificar conceptos, ideas y relacionarlas con ideas anteriores, conceptos que casi nunca son el nombre de una sensación. Cuando un cerebro es capaz de clasificar ideas, agruparlas, relacionarlas y crear nuevas clases a partir de clases anteriores decimos que estamos ante una persona *inteligente*. En el hecho anterior se basan algunos exámenes de coeficiente intelectual. Una persona será más inteligente mientras más relaciones establezca y mas clasificaciones haga[1].

Existen por otro lado, tareas que son difíciles desde el punto de vista humano; saber exáctamente qué tono tiene una nota musical, distinguir un tono de gris de otro tono

---

1. En el test de dominós cada pregunta consiste en la presentación de un grupo de 4 dominós que tienen una cierta secuencia y responder es localizar el dominó que completa la secuencia. El dominó se elige de un conjunto de 5 que se presentan como alternativas. Traducido al lenguaje del reconocimiento de patrones significa que el dominó que falta pertenece a la clase definida por los dominós dados como ejemplo. Los otros cuatro de la respuesta no pertenecen a la clase.

muy cercano. Son labores que realizan los expertos; músicos con oído absoluto, pintores etc. Una computadora, con la instrumentación adecuada, puede realizar estas labores de manera muy precisa; sin embargo clasificar no es un problema que se pueda tratar fácilmente, de un modo generalizado, con una computadora; es un problema abierto. Existen técnicas que permiten resolver parcialmente algunos problemas, reconocer texto impreso, palabras habladas de un vocabulario limitado y otros.

Queremos hacer incapié en la complementaridad que representa el uso de computadoras. Tareas que son fáciles para una máquina (y podemos citar ejemplos no triviales como probar teoremas y resolver ecuaciones algebráicas) resultan descomunales para un ser humano sin el entrenamiento adecuado. Paralelamente, resolver problemas sencillos, hasta cotidianos para un ser humano, envuelve un trabajo muy complejo para una computadora. El problema de fondo que se observa, reside en la falta de comprensión actual de los procesos mentales; hace algunos años se creía imposible construir una máquina para jugar ajedrez, hoy existen programas que pueden jugar a un nivel aceptable y son resultado de una mejor comprensión del juego de ajedrez. Resolver el problema de la percepción significa entender, a nuestro juicio, qué es lo que sucede cuando percibimos; sin embargo, la formalización no es garantía de solución, como podemos observar en los teoremas que limitan la eficacia de los sistemas formales.

Ahondando en esta discusión, quisieramos hacer notar la falta de conocimiento del funcionamiento del cerebro con un ejemplo sencillo de teoría de la computación:

Definamos un algoritmo como una máquina finita que procesa una cadena de entrada para producir una cadena de salida en cierto sistema formal. A esta máquina finita o algoritmo le podemos asociar un número que llamaremos **complejidad** y que consiste en medir el número de operaciones elementales que se realizan para procesar la cadena de entrada. La complejidad estará dada como una función de la longitud de la cadena de entrada, de esta manera tendremos algoritmos de complejidad lineal, polinomial, exponencial, etc.

Resulta que problemas como el cálculo de predicados, si bien es formalizable, tiene una complejidad exponencial; lo que significa que para cadenas de tamaño 100 emplearíamos una cantidad de tiempo impensablemente grande para resolver el problema (del orden de la edad del universo) aún si realizamos una operación elemental cada millonésima de segundo. Pese a esta limitante, podemos observar que un estudiante de licenciatura es capaz de resolver "instantáneamente" una expresión para realizar un cálculo en este sistema formal.

Otro ejemplo interesante consiste en el problema de traducción automática. Según los resultados de Noam Chomsky, el lenguaje natural se puede expresar en términos de un sistema formal que no tiene un algoritmo polinomial de resolución ¡y en la vida real existen políglotas!

De la discusión anterior, podemos afirmar que la formalización de un problema es apenas la primera etapa en la búsqueda de soluciones realistas. De ello se nutre la discusión de la búsqueda de otras arquitecturas para la resolución de problemas, entre ellas, la de máquinas de procesamiento paralelo, redes neuronales artificiales y algoritmos genéticos, todos ellos enfocados a la búsqueda de algoritmos que emulen la eficiencia de la mente humana.

1.2 La comunicación hombre-maquina

Desde la invención de la primera computadora hasta la masificación de su uso en nuestros días se ha construido una pirámide de programas, metodologías y técnicas que tienden a facilitar el uso de las mismas en la resolución de problemas. En un principio era necesario modificar el esquema de pensamiento para acomodarlo a la rígida estructura de un lenguaje meramente numérico cuando se quería resolver un problema. Esta tendencia evolucionó cuando se diseñaron los primeros lenguajes de programación hasta llegar a los llamados sistemas "front-end", que consisten en programas de construcción muy compleja cuyo uso es relativamente sencillo.

A pesar de que los sistemas de propósito general citados arriba son de manejo muy simple, los usuarios de estos sistemas tienen que pasar por un entrenamiento mínimo

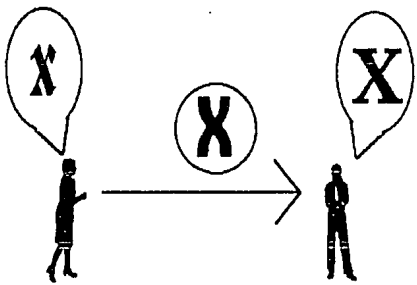


figura 1
El proceso de comunicación
Las ideas concebidas son modificadas al transmitirse y al recibirse; esto significa que la comunicación es imposible o por lo menos no formalizable.

para utilizarlos. El uso de periféricos "intuitivos" y de interfaces gráficas ha permitido simplificar aún mas el uso de la computadora; sin embargo se sigue requiriendo de una adecuación de los esquemas de pensamiento, pensar dentro de marcos fijos y en cierta medida artificiales, para poder utilizar una computadora.

Yoh-Han Pao[1] establece un origen común de la inteligencia artificial y el reconocimiento de patrones; este origen está localizado en la búsqueda de un relajamiento de los estrictos esquemas de pensamiento discutidos en los párrafos anteriores. El reconocimiento de patrones y la inteligencia artificial tendrían como objetivo la construcción de interfaces amigables para el uso de las computadoras. Estas interfaces deberán permitir la interacción con las máquinas en un marco natural e intuitivo para los seres humanos. Para enriquecer esta idea sometamos a análisis el proceso de pensamiento que se sigue cuando se comunican dos personas[2]:

Si A tiene una idea que desea comunicar a B, elabora la idea de acuerdo a su propio esquema de pensamiento; pero transmite un mensaje que puede ser incompleto o contra-

dictorio. B recibe el mensaje y lo interpreta de acuerdo a su propia ideosincrasia, cuando le comunica el "acuse de recibo" a A nuevamente, la idea que transmite es modificada otra vez; sin embargo A deduce que ¡el mensaje fué correctamente entendido por B!. (fig. 1)
Este nudo de incomunicación es inherente al ser humano. El único método para eliminar esta paradoja es el utilizar un lenguaje formal como el de la matemática; pero desde luego que no podemos comunicar todas nuestras ideas mediante un lenguaje tan estricto como éste; las ideas estarían restringidas a un pequeño universo construible con los símbolos del lenguaje.
Cuando nos comunicamos con una computadora (sea lo que fuere que significa esto), restringimos nuestro universo de ideas al tratar de plasmarlas en el lenguaje formal (lenguaje de programación) que acepta una máquina. No podemos "decir" todo lo que podemos expresar en un lenguaje natural. Este problema tiene dos orígenes: 1)- la máquina no puede hacer todo lo que se nos ocurre y 2)- el proceso de formalización de la idea de una tarea que la máquina puede ejecutar puede ser tan complejo que se necesite de un especialista para elucidarlo.
A nuestro juicio y conviniendo con algunas de las ideas de Pao, la inteligencia artificial se encarga de resolver este vacío en la comunicación hombre-máquina. El reconocimiento de patrones sería el sustrato básico para construir una interface inteligente, sería la capa elemental o de bajo nivel; la inteligencia artificial tendrá como objetivo la mecanización del proceso de formalización, de tal sucrte que se pueda interaccionar con la máquina de un modo semejante al que interaccionamos con otros seres humanos.

1.3 El problema de reconocimiento de patrones

Zadeh[3] refiere el problema de reconocimiento de patrones como la aplicación de un mapeo opaco y un mapeo transparente. El mapeo opaco es el utilizado por un ser humano al establecer relaciones entre objetos, al clasifi-

carlos y asignarles propiedades inferidas de las características evidentes (una de las cuales sería la clase a la que pertenece). El mapeo es opaco porque la persona no tiene claro porqué puede distinguir y agrupar objetos, simplemente es capaz de ello. Cuando vemos la espalda de un conocido en una multitud, podemos deducir el "resto" de la persona; sin embargo no somos capaces de establecer (sin pensarlo mucho) el mecanismo mediante el cual hicimos esa deducción. La persona que establece las relaciones no tiene claro el modo en que las realiza. La tarea de un especialista en reconocimiento de patrones consiste en proponer un mapeo transparente que resuelva el mismo problema.

King Su Fu[2] sugiere que existen dos mecanismos para la realización del reconocimiento de patrones. El primero de ellos es la abducción, que consiste en emitir hipótesis válidas de un conjunto de hechos. Estas hipótesis registrarán la estructura del sistema en el que nos encontramos inmersos (dicho de otra manera, la elaboración de las leyes que lo rigen). La inducción, consiste en probar si una ley dada se cumple en un universo particular. La inducción se puede dar de manera mecánica; la abducción involucra un proceso inteligente.

Para redondear el marco general en el que versará nuestra discusión, conviene señalar un aspecto sugerido en el artículo de Pao, una de las características mas fuertes de los sistemas de inferencia (abducción) consiste en su adaptabilidad; las reglas inferidas del sistema pueden modificarse en presencia de contraejemplos a una generalización dada. Esto provoca una relajación (cuando se puede) de la regla en análisis, lo que da como resultado una generalización. Por el contrario, cuando eliminamos uno de los ejemplos que condujeron a enunciar la regla, se produce una especialización.

Con lo discutido en la sección anterior, hemos establecido un espíritu de procedimiento que debe ser plasmado en una implantación física de las observaciones y consideraciones generales; mas aún cuando se trata de una instancia del problema general. Discutiremos en esta sección el problema que nos ocupa en este trabajo (reconocimiento

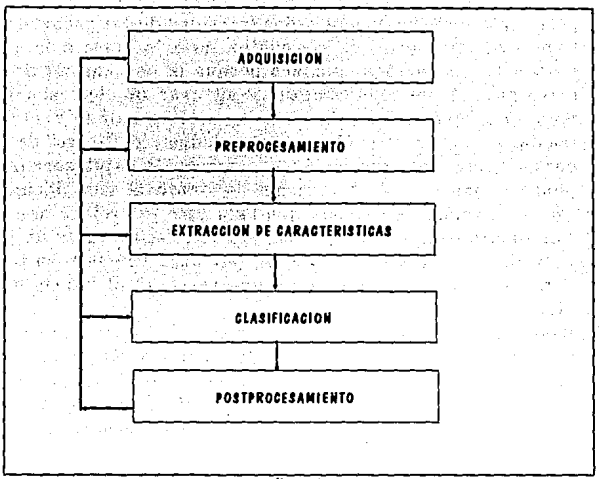


figura 2
Etapas del reconocimiento de patrones
El significado de las flechas retroalimentación que puede ocurrir en cualquier etapa

optico de caracteres), orientando la visión hacia el problema general que no se circunscribe a este ámbito.

1.4 Formalización

Dada la obligada referencia a sistemas reales de reconocimiento de patrones, que van desde los elementales ojos simples de una mosca que pueden distinguir con precisión cuando están a punto de darle un manotazo o un molusco que puede reconocer lo que es comida de lo que no es y reaccionar en consecuencia, es necesario hacer la aclaración de que la discusión siguiente tiene un ámbito de aplicación muy restringido y obedece a *un modelo independiente* o artificial del proceso de reconocimiento. Este modelo es construido con base en las herramientas matemáticas disponibles y refleja la concepción formal del problema; no tiene bases fisiológicas y corresponde al mapeo transparente que discute Zadhe.

La figura 2 representa el proceso general para el reconocimiento de patrones; formado por cinco módulos básicos, *adquisición, preprocesamiento, extracción de características, clasificación y postprocesamiento.*
Discutiremos brevemente cada uno de ellos.

adquisición

El primer módulo consta de un arreglo físico de sensores que transforman una escena en un arreglo numérico; este arreglo es usualmente de dimensión muy alta y contiene la digitización de las señales que forman los objetos con los que se trabajará. Si fuera reconocimiento de voz, las señales serían digitizaciones de ondas sonoras; en este caso, como trabajamos con caracteres impresos la escena es una imagen binaria que entrega un scanner.
La explicación detallada del funcionamiento de un dispositivo de adquisición está mas allá de la intención de este trabajo. La construcción de un dispositivo de adquisición es mejor explicada en un texto de electrónica digital y de procesamiento de señales.

preprocesamiento

En la obtención del arreglo numérico vamos a tener *ruido*, este es un concepto tomado de la teoría de la información, que en la connotación específica del reconocimiento de patrones significa la concurrencia de errores dados en circunstancias no ideales durante la obtención de los objetos como representaciones numéricas. Estos errores pueden ser de naturaleza variada; algunas posibles fuentes de ruido en esta etapa son: errores de medición, distorsiones del objeto al presentarse al digitizador en una forma no canónica, variaciones en la alimentación de voltaje al sistema, influencias externas (electromagnéticas, suciedad en un lente) etc.
Esta es una etapa de bajo nivel, generalmente no tomada en cuenta en la literatura de reconocimiento de patrones. El dominio de esta etapa está localizado en la electrónica digital, la física del estado sólido y campos afines; el interés principal de esta etapa, consiste en presentar una imagen numérica fiel a la imagen real que acota el digitizador.

Dado que habrá ruido intrínseco al obtener la imagen del digitizador, el diseñador de este dispositivo debe proporcionar información acerca del comportamiento estadístico de este ruido; deben quedar establecidos los límites de sensibilidad, la resolución, la granularidad y la estabilidad del dispositivo. Esta información servirá para modelar el ruido que puede generar esta etapa.

En el segundo módulo, vamos a aplicarle un tratamiento a la escena obtenida; se tratará de reducir el ruido que se introdujo en la obtención y se segmentará la escena de tal forma que podamos identificar los objetos que la forman y discriminarlos del fondo. Esta es otra fuente de ruido; podemos identificar posibles errores como un filtraje inadecuado o una segmentación inapropiada.

extracción de características

Los objetos segmentados representan toda la información que podemos obtener de la escena con el digitizador empleado: no hay otra fuente de información y debemos utilizarla para distinguir al objeto. Usualmente tenemos como resultado de la digitización un arreglo numérico de dimensión muy alta (p/ej. una imagen digital o una digitización de voz) que resulta intratable para fines prácticos. Debemos entonces proceder a reducir la dimensionalidad de este arreglo, o dicho de otra forma a concentrar la información. Si tuvieramos a la mano un método perfecto de compactación de la información el resultado de la extracción de características sería un número único para cada clase de objetos y la clasificación sería trivial.

La tarea del tercer módulo es precísamente la compactación de la información que es conocida como extracción de características. Desde el punto de vista del álgebra lineal, lo que estamos haciendo es una proyección de un espacio m dimensional sobre uno de dimensión n menor que m. Esta proyección puede ser descrita en términos de una receta ad hoc o por el contrario puede ser construida en base a un análisis estadístico de un conjunto de muestra etiquetado con la clase a la que pertenece cada elemento.

La construcción ad hoc de la proyección es el método mas socorrido en aplicaciones en donde la estructura de los ob-

jetos es conocida por un experto; este conocimiento se plasma como una transformación matemática que aplicada a los objetos digitizados produce un vector que refleja la ausencia o presencia de propiedades definidas por el experto. Cada entrada del vector de características representa la medición de alguna propiedad descrita primero en términos verbales y formalizada después como una transformación sobre el arreglo numérico.

El otro tipo de proyección se realiza sin tomar en cuenta conocimiento a priori, es llamado *extracción automática de características* y se refiere a la obtención de una proyección lineal (en el sentido de que las nuevas coordenadas son combinaciones lineales de las coordenadas anteriores) que produzca clusters densos en el nuevo espacio. Toda la información para la transformación es obtenida de las estadísticas de los objetos muestra.

clasificación

El clasificador es un método para la identificación de los objetos sometidos a discriminación. Con algún grado de certeza (determinado por el método de clasificación empleado) se determina la clase a la que pertenece el objeto a clasificar. Este módulo tiene dos vistas; en la primera debemos hacer "entrenamiento", i.e. determinación de parámetros estadísticos y geométricos intrínsecos a cada problema de clasificación. Estos parámetros determinan regiones en el espacio n-dimensional en donde se agruparán las distintas clases de objetos a clasificar. Este entrenamiento puede ser supervisado, no supervisado o una mezcla de ambos. La segunda vista de este módulo es la clasificación propiamente dicha, la cual deberá hacerse en base a los parámetros determinados en la vista anterior. Básicamente consiste en localizar en qué región o partición está localizado el objeto (un punto en el espacio) y determinar qué etiqueta tiene esa región.

postprocesamiento

Una vez digitizado, segmentado y clasificado el texto pueden existir errores que no fueron detectados. El postprocesamiento consiste en la validación contextual de los objetos que fueron clasificados en base a información de alto

nivel que se obtiene al interpretar la escena que fue procesada. Se utiliza una interpretación jerárquica de la escena para construir una validación de los elementos que la forman. El uso más claro para técnicas de este estilo es en el problema de reconocimiento de texto; un conjunto de caracteres separados por espacios forman palabras, las palabras forman frases y las frases forman párrafos etc. Los sistemas de postprocesamiento son construidos de tal modo que se busca que los objetos individuales tengan sentido para la jerarquía siguiente.

1.5 Discusión del trabajo realizado

El objetivo de este trabajo es presentar una introducción a los temas básicos de reconocimiento de patrones, tomando como caso de estudio el reconocimiento óptico de caracteres. Se selecciona este subtema del reconocimiento óptico de caracteres (OCR por sus siglas en inglés) por la inmediatez en la obtención de ejemplos concretos para el análisis. Buscaremos sustentar la idea de un clasificador multiescala, discutido con detalle una vez expuestas las generalidades del tema, que es la contribución de este trabajo.

Para la realización del objetivo planteado, para redondear las ideas y al mismo tiempo tener elementos de juicio en la selección de un método particular de trabajo en una implantación realista de OCR (en el caso de que se pensara en la construcción de un sistema para venta al público), se construyeron los módulos descritos en el esquema de la figura 2. Discutimos los alcances de los módulos implementados y en cada uno se describen las rutinas principales.

adquisición:

Consiste de un conjunto de rutinas que leen una imagen binaria obtenida de un scanner y entregada en el formato TIFF. Dado que la cantidad de memoria que se necesita para una imagen binaria de una página completa de texto es muy alta, se construyó un manejador de memoria en el formato EMS de las computadoras compatibles con IBM. Este manejador provee un solo buffer de tamaño arbitrario (tanto como memoria alta tenga la computadora) y

se proveen rutinas para manejar el buffer en memoria baja.

Se construyeron además un conjunto de rutinas para escribir un buffer (una imagen binaria) en formato TIFF, esto obligado por la necesidad de generar imágenes impresas que ilustraran las etapas de resolución del problema.

rutinas principales:

void rd_tiff(ImgType *im, char *buffer_name)

void wr_tiff(ImgType *im, char *buffer_name)

Leen o escriben respectivamente las imágenes que se encuentran en el disco con el nombre buffer_name y la colocan o toman de im

preprocesamiento

Son un conjunto de rutinas que realizan un filtraje de media en un buffer en memoria para la remoción de puntos aislados. Funcionan con templetes que se mueven en la imagen y son discutidos en el capítulo correspondiente.

Se construyeron rutinas de segmentación para texto presentado en forma canónica i.e. con los renglones coincidiendo con las líneas horizontales de la figura. No hay rutinas de corrección de orientación del texto (rotación), ni de detección de la orientación de la página.

Contiene también rutinas para filtrar letras individuales una vez que son segmentadas. La idea es binarizar las letras para poder utilizar la distancia de hamming en el clasificador; pero se requiere que estas no presenten demasiadas variaciones debido al contraste para reducir el número de ejemplos en la base de datos. Todo esto se detalla en el capítulo correspondiente.

rutinas principales:

ImgType *pnt_clr(ImgType *im, char grano)

Limpia la imagen pasada en im de puntos aislados de grano bits o menos.

Block_Type *G_Block(ImgType *im)

Regresa un apuntador a una lista ligada que contiene en cada nodo las coordenadas de inicio y fin de los renglones(bloques de texto)

CarType *G_char(ImgType *im, BlockType *b)

regresa una lista ligada con las coordenadas de cada uno los caracteres encontrados en los renglones de la imagen. En caso de encontrar un

fin de linea o un espacio en blanco inserta un nodo marcado como caracter especial para no procesarlo.

extracción de características

No se utilizaron definiciones adicionales de "features" para la clasificación, prácticamente consiste en un escalamiento para colocar las letras en una rejilla con un número fijo de cuadros. La rutina recibe una letra segmentada y entrega un vector que contiene los promedios de gris en cada cuadro de la rejilla. No hay propiamente una extracción de características.

rutinas principales:

float *v(CarType *car, unsigned char m, unsigned char n)

Regresa un vector de dimensión m*n que contiene los promedios de gris en cada uno de los cuadros de la rejilla que se superpone al caracter dado en car.

char *FloatXbin(float *V, unsigned char m, unsigned char n)

transforma el vector dado en V en un vector binario representado en una matriz de bits, utiliza un filtraje de media para obtener los pixels.

clasificación

Se implementaron dos clasificadores: la máquina lineal de Fisher y el clasificador de *k*-vecinos. El primero utilizando la distancia euclideana y el segundo con la distancia de hamming.

Hay adicionalmente la generación de una lista de alternativas de clasificación en el caso de empates, el primer elemento de la lista es el que tiene mas posibilidades, y en orden decreciente se colocan los que corresponden a clasificaciones alternativas.

No se construyó la etapa de retroalimentación que permite segmentaciones alternativas con base en la información contextual del postprocesamiento.

rutinas principales:

/* distancia euclideana */

CharTable *AddChar(char id, CharTable t, float *v)

Actualiza los parámetros del prototipo de clase identificado con id, utilizando la información que

se le pase en v. Básicamente recalcula la media usando la expresión iterativa de ella. t contiene la dimensión de clasificación que se está usando.

CharList *Get_Char(CharTable t, float *v)

mide la distancia del vector v a cada uno de los elementos de la lista ligada t y regresa una lista de identificadores que corresponden a los prototipos que tienen distancia menor al radio de la clase (en porcentaje) en orden descendiente.

/* distancia de hamming */

Char_H_Table *AddChar(char id, Char_H_Table t, char *v)

Busca en la tabla t la entrada que corresponde al identificador id, si existe un vector igual al que se le manda incrementa el contador de duplicidad de ese representante; si no hay ninguno igual crea un nuevo nodo y pone el contador de duplicidad en 1.

char Get_H_Char(Char_H_Table t, float *v)

mide la distancia del vector v a cada uno de los elementos de la tabla t y regresa el identificador vector que tuvo votación mas alta. En caso de empate el caracter se rechaza.

postprocesamiento

Se implementaron rutinas de búsqueda de palabras en un diccionario con el método de hash externo. Para optimizar la búsqueda de palabras, se construyó un autómata finito de secuencias que detecta palabras "razonables" antes de consultar en la tabla del diccionario. El resultado de esta etapa es un rechazo o una aceptación de cada una de las palabras separadas por espacios que se le presentan en un archivo; si la palabra no existe en el diccionario se le pide al usuario que teclee la palabra correcta, si la palabra contiene un caracter rechazado no se busca en el diccionario y se presenta directamente.

rutinas principales:

char preview(automata *m, char *word)

regresa true si el autómata m reconoce la palabra dada en word o false si no la reconoce. Si regresa true aún hay que intentar localizarla en el diccionario, si regresa false es seguro que no se encuentra en el diccionario.

unsigned int hash(char *word)

regresa el número de hash de la palabra word; este

número se utiliza para localizar en la tabla interna la dirección física del archivo handle.

char G_word(int hash, Hash_Table t, int handle, char *word)

regresa true si en la dirección física del archivo handle se localiza la palabra word. En caso de colisiones en la tabla t intenta todas las direcciones que se encuentren en ese cluster.

El sistema no funciona para usuarios finales, es un prototipo que fué útil para la elaboración de este trabajo y no tiene sino objetivos académicos.

2

PREPROCESAMIENTO

El resultado final del preprocesamiento consiste de una lista de los elementos que forman la escena. El primer paso para obtener la lista consiste en eliminar el ruido que pudiera contener la digitización; este es un punto delicado. La definición de ruido es elusiva y confusa cuando no poseemos información detallada del comportamiento estructural de los pixels que forman la imagen; en general podemos afirmar que el ruido es una perturbacion a una estructura jerárquica subyacente en la escena i.e. si estamos analizando pixels el ruido son pixels con información errónea, si son bloques de pixels el ruido serán manchas o agregaciones de pixels con una estructura no canónica etc. En el caso particular que nos ocupa, tratamos con pixels mal colocados. Discutimos brevemente el método seguido para localizar los pixels mal colocados en la imagen.

2.1 Eliminación de ruido

Cuando aplicamos un digitizador (scanner) a una hoja de texto, obtenemos un mapa de pixels que representan las intensidades relativas de puntos discretos en la página. Este mapa de pixels puede ser binario (prendido o apagado) o puede tomar valores en un intervalo de 0 a 255, donde 0 significa negro y 255 significa blanco. Escogeremos una digitización binaria, los puntos en la hoja de texto que sobrepasen un cierto valor de contraste, serán considerados como 1 y los que queden bajo ese valor como cero. Suponemos la representación de la hoja de texto como una matriz de bits que podemos accesar en un arreglo.

Dado que los pixels que se encuentran en la imagen solo

pueden tener dos valores necesitamos información contextual para decidir si un pixel particular tiene el valor adecuado. Esta información la obtendremos de los pixels vecinos y habrá una votación para decidir si el centro de un templete colocado sobre cada pixel se debe invertir.

El templete es una matriz t de 3x3; la imagen es una matriz p, y los pixels los elementos p_{ij} de la matriz. Para cada p_{ij} que no se localize en la frontera debemos obtener la suma

(2-1)

$$s = \sum_{k=-1,l=-1}^{k=1,l=1} [p_{i-k\,j-l}][t_{k+2\,l+2}] \qquad t=\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$$

esta es una manera de contar los pixels que se encuentran en la vecindad de p_{ij} y el máximo valor de s es 8 y el mínimo 0. Con esta información debemos decidir que hacer con el pixel central. La decisión que hay que tomar es si debemos cambiar el pixel central dependiendo del número de pixels encendidos en la vecindad y del valor del centro mismo. Tenemos para ello 2^{10} posibilidades; analizar cada una por separado representaría demasiado trabajo y no sería un método razonable. En aras de simplificar el análisis, utilizaremos una nomenclatura diseñada por nosotros en donde cada regla está representada por una cadena de combinaciones de 4 posibles símbolos (0, 1, *, #) y de longitud nueve. El primer símbolo de la cadena significa qué hacer cuando el templete encuentre una suma de 0, el segundo cuando la suma sea 1, etc. El significado de cada símbolo es

0 colocar un cero en el centro sin importar el valor anterior
1 colocar un uno en el centro sin importar el valor anterior
***** dejar el centro sin cambio
invertir el valor del centro

de esta forma, la cadena 0000*1111 significa: para sumas menores que 4 colocar un cero en el centro y para sumas mayores que 4 colocar un 1.

Se realizaron diversos experimentos para determinar que

reglas producen un filtraje adecuado, encontrando los resultados sumarizados en las figuras.

Discutimos brevemente los resultados obtenidos. Suponemos que las letras tienen color negro (0) y el fondo tiene color blanco (1).

La parte derecha de la cadena (los últimos cuatro caracteres) localiza puntos en el fondo, ruido fuera de las letras. La parte izquierda localiza ruido en las letras. El centro es el caso de empate; dado que estamos trabajando con los vecinos de un pixel, tenemos solo información local, y la única alternativa es dejar el centro sin cambio.

La regla 0*******1 remueve puntos aislados sin tocar las fronteras de las letras; de manera simultánea remueve puntos y pares de puntos en el fondo y en las letras.

La regla 00*****11 remueve agrupaciones de tres pixels. Modifica las esquinas de las letras y algunos puntos en donde la frontera no está bien definida. Después de algunas iteraciones se estabiliza y deja sin cambio la imagen.

La regla 000***111 remueve agrupaciones de hasta cuatro pixels, adelgaza las letras pequeñas y adornos no conexos (como el punto de la i o j) se estabiliza después de algunas iteraciones pero hace desaparecer letras que no tienen un grosor mínimo.

La regla 0000*1111 remueve agrupaciones de hasta 5 pixels, pero modeifica el contorno de las letras dependiendo de la rugosidad de las fronteras.

En general la aplicación de reglas que de manera simultánea remueven ruido en el fondo y en las letras no es conveniente; porque se llega a moficar sensiblemente el contorno de las letras. Colocar un cero o un uno en el caso de empate (en el centro) adelgaza o engorda las letras.

Las figuras 1 y 2 muestra un filtrado basado en la aplicación de varias reglas.

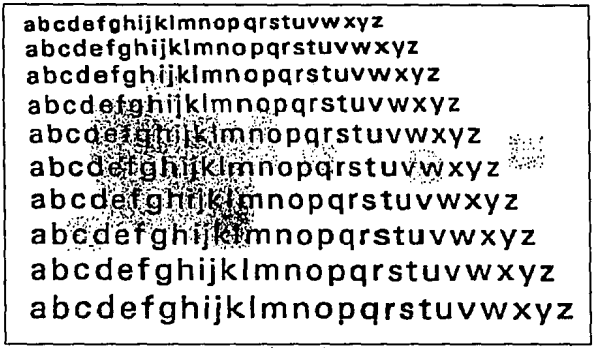
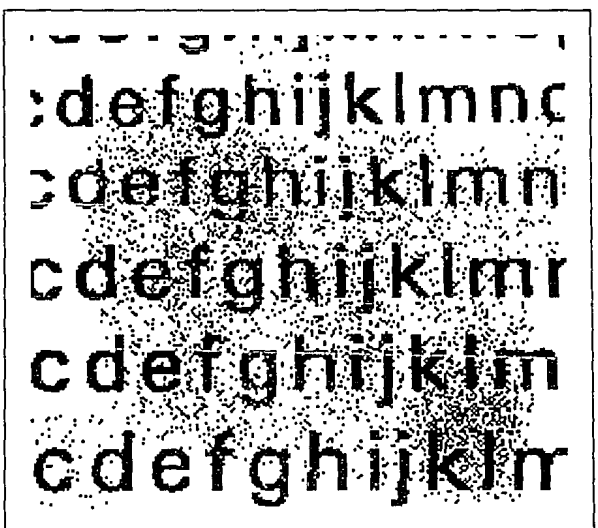


figura 2.1

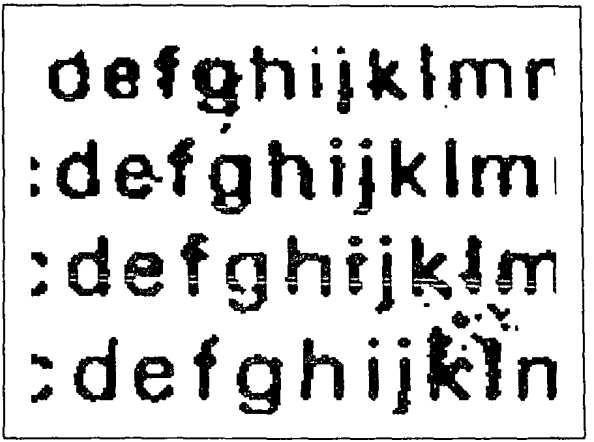
Imagen original con ruido

A esta figura se le agregó ruido en la región que se muestra. El ruido se generó en una ventana de tamaño 42 pixels, en cada ventana se agregaron 20 pixels con distribución uniforme. La ventana se aplicó repetidamente en regiones aleatorias dentro de la zona de la mancha



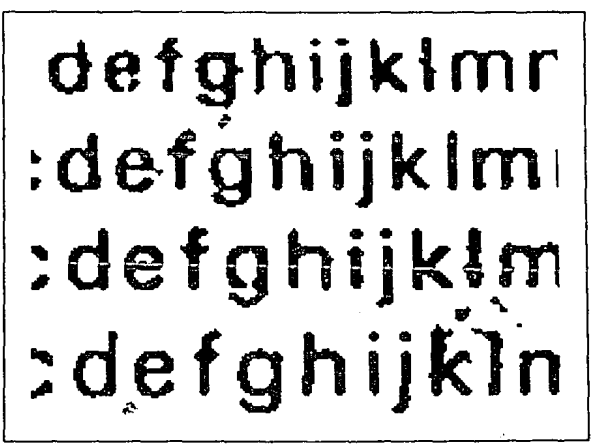
2.2

Un acercamiento a la mancha



2.3

Filtraje con la aplicación de la regla 0000*1111, con tres iteraciones. Obsérvese como se degradan las fronteras de las letras.



2.4

En este filtraje alternamos diversas reglas. 4 iteraciones con ******111, seguido de una iteración con ****1111 y una con 0000****.

2.2 segmentación

La segmentación juega un papel crucial en el desarrollo de clasificadores de patrones, es la etapa mas crítica. Esto significa que errores cometidos en la segmentación no podrán ser corregidos al extraer características o clasificar y solo serán detectados al posprocesar el texto. Los sistemas comerciales de OCR tienen este talón de aquiles; en ellos se recomienda al usuario que digitize los textos a una resolución alta y un contraste bajo, con el afán de permitir que las letras queden perfectamente separadas.

Discutiremos en esta sección los problemas típicos que se presentan cuando se quiere segmentar la imagen, y sugeriremos algunas técnicas que se utilizan.

El problema de segmentar letras tiene connotaciones muy sofisticadas, sobre todo cuando las letras se hallan inmersas en una escena compleja que contenga paisajes, letras de distinto tamaño, cuando no hay párrafos definidos etc. Estas situaciones rebasan por mucho las espectativas de la investigación de punta en esta rama; es difícil ubicar una pronta resolución de este problema, ya que no hay herramienta teórica poderosa que permita el análisis de escenas complejas, ni desde el punto de vista meramente computacional ni pensando en niveles fisiológicos. Nos remitiremos pues a texto plano como el que se encuentra en una hoja común de texto en un libro sin figuras ni recuadros (del tipo de un periódico). Toda la discusión estará basada en este supuesto.

descripción del problema

La hojas de texto impresas están formadas por una jerarquía de bloques, a saber, los párrafos, líneas, palabras y letras. El objeto sobre el que hacemos clasificación son las letras y el fin último es alojarlas en rectángulos que las contengan exáctamente. La localización de cada uno de los elementos de la jerarquía (párrafos, renglones, etc.) es de utilidad cuando vamos a postprocesar el texto.

Líneas de texto

La curva formada por puntos discretos mas sencilla de enunciar y programar es la línea recta y dentro de las rectas, aquellas que tienen pendiente infinito o cero. Este he-

cho da origen a que se busque separar las letras mediante rectas, es decir, colocarlas dentro de un recinto rectangular que contenga exactamente un caracter.

La localización de estos recintos rectangulares no siempre es posible de manera inmediata. Cuando el texto se encuentra rotado algunos grados con respecto a los ejes del dispositivo de adquisición, no es posible adecuar rectas de pendiente cero o infinito para separar bloques; de ahí que cuando el texto está rotado sea necesario encontrar el ángulo de inclinación de la hoja con respecto al scanner, esto debe hacerse de manera automática y representa una de las operaciones mas costosas, en tiempo de máquina, que se hacen a las imágenes digitales; dicho de otra manera, este factor puede hacer disminuir sensiblemente la velocidad de reconocimiento en cualquier sistema de OCR.

Palabras y letras

Las palabras de un texto son fácilmente identificables dado el espacio entre palabras. Es difícil encontrar ejemplos de palabras que no se puedan segmentar. Debemos marcar las coordenadas de inicio y fin de cada una de las palabras para la etapa de postprocesamiento.

Dentro de cada palabra, una vez segmentada, debemos localizar con la mayor exactitud cada una de las letras que la forman. Dividimos en cuatro los casos de separación entre letras, estas son: *separadas*, *traslapadas*, *pegadas y partidas*.

Las primeras corresponden a pares de letras que podemos separarar por medio de una recta paralela al eje *y*, es el caso mas fácil. Los pares pueden estar pegados, i.e. las dos letras comparten pixels o sin compartir pixels no se puede hacer pasar una recta paralela al eje *y*. Cuando la calidad del texto no es buena, el contraste al que se digitizó la letra es muy bajo o cuando las letras son obtenidas de una impresión de matriz de puntos, tendremos el caso de letras partidas, es decir, no existe una región de pixels contiguos que formen la letra; por el contrario encontramos que la letra está formada por un conjunto de pedazos o islas de pixels que no son contiguos.

La discusión anterior ilustra cuales son los casos de inte-

rés en la segmentación de las letras; pero no auxilia en la misma. Podemos ver bloques de texto segmentado sin error hasta el nivel de palabras; mas allá de eso desconocemos cual es el caso de los bloques que estamos segmentando: estos bloques pueden ser letras perfectamente separadas, pedazos de letras o grupos de dos o más letras. Necesitamos un procedimiento inteligente (en el sentido de un sistema experto) que determine con algo más que un 50% de probabilidad que el bloque que estamos aislando es en efecto una letra.

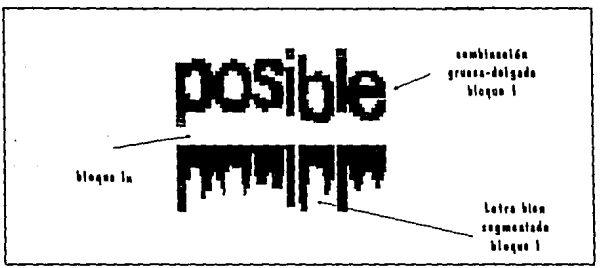
Algunas estadísticas acerca de la hoja de texto son de utilidad para determinar si lo segmentado es una letra. Estas medidas son

1 Ancho promedio de los bloques candidatos a letras
2 Densidad promedio de los bloques (número total de pixels encendidos en cada bloque entre área total ocupada)
3 Altura de la línea base con respecto del bloque de renglones
4 Ancho promedio en un bloque correspondiente a una palabra
5 Gráfica de densidad vertical de las letras

El procedimiento para segmentar las letras quedaría integrado así:

1 Localizar todos los bloques horizontales, bloques h.
2 Obtener el espacio promedio entre bloques. (esto nos da una medida a priori del espacio que esperamos tener entre palabras)
3 Segmentar las palabras de cada bloque usando el dato anterior y localizando los espacios grandes entre letras para limitar palabras, bloques p.
4 Dentro de cada palabra localizar los bloques candidatos a letras, bloques l. En este proceso elaboramos una gráfica de densidad vertical de las letras.
5 Obtener el ancho promedio de los bloques l
6 Marcar todos aquellos bloques que tengan ancho mayor que el ancho promedio de los bloques l. (bloques l_M)

2.5
Gráfica de densidad vertical de las letras.



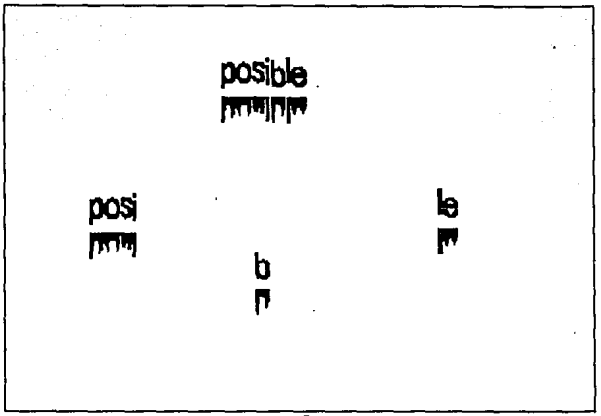
2.6
Ocurrencias de los casos I_M y I al segmentar.

7 Marcar todos los bloques que tengan ancho menor que el ancho promedio de los bloques I. (bloques I_m)

Una vez marcados los bloques I_M y I_m, decidimos si se trata de grupos de letras o segmentos de letras.

8 Para todos los bloques I_M, si se puede localizar una densidad vertical pequeña alrededor de la región del ancho promedio, los dividimos en los que sean necesarios (si el bloque tiene ancho 3*ancho promedio, intentaremos dividirlo en tres bloques, si tiene ancho 2*ancho promedio en dos bloques, y asi para los demás casos) colocándolos en una lista de segmentación alternativa.

9 Si hay pares de bloques I_m contiguos, sospechamos que son una sola letra (habrá errores en los casos -ii- -li- etc. cuando haya dos letras delgadas juntas) y agregamos a la lista de segmentación alternativa.



2.7
Segmentación sin utilizar información de la gráfica de densidad vertical. Como es claro, existen errores de segmentación debido a letras pegadas.

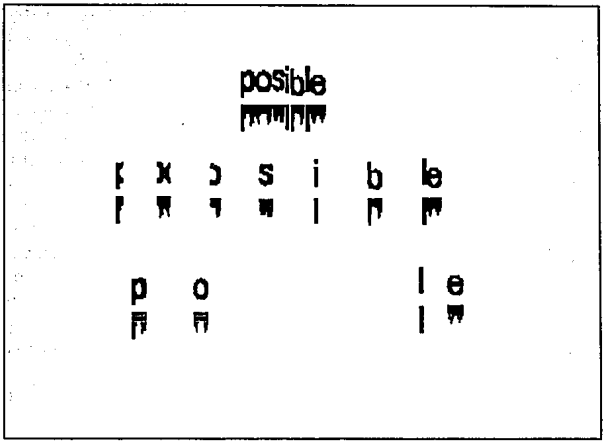
Al final de este procedimiento tendremos una lista de los bloques candidatos a letras, junto con una lista de segmentación alternativa en los casos en donde haya duda.

La altura con respecto a la línea base es un dato que se pasará al clasificador como parte de la letra y servirá como elemento de arbitraje cuando haya confusión en la clasificación. Si encontramos un punto que está muy arriba de la línea base, se trata posiblemente de ruido.

Cada palabra segmentada tendrá su propia estadística de ancho promedio; si cada bloque l dentro de un p tiene un ancho semejante a la estadística global (dentro de ciertos márgenes) procesaremos el texto usualmente; pero si un bloque l es anormalmente ancho habrá mas razones para considerar que se trata de un grupo de letras.

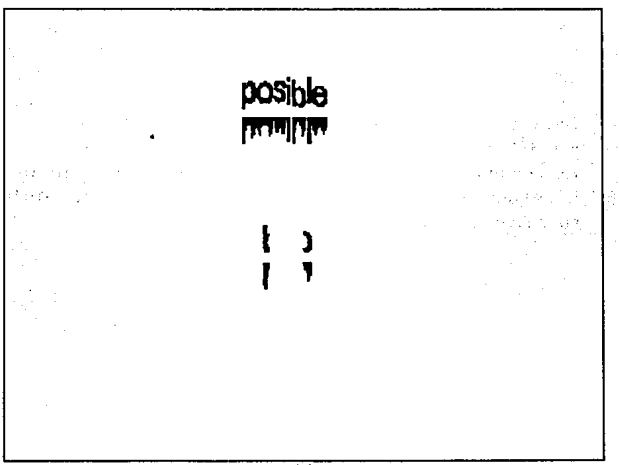
La gráfica de densidad vertical en cada bloque de palabra determinará las regiones en donde se puede hacer una segmentación alternativa; donde haya un bloque lM en conjunción con una región delgada en la gráfica se segmentará en esa región. [fig. 2.8]

Una combinación difícil de tratar será la que tenga pares



2.8

Segmentación utilizando la gráfica de densidad vertical. Nótese que tenemos varias alternativas, las cuales se agrupan en una lista. Esta lista se utilizará al tiempo de clasificación.



2.9

Si se utiliza la gráfica de densidad vertical sin retroalimentación en la etapa de clasificación no es posible distinguir entre un par delgada-delgada y una letra de ancho normal.

de letras delgadas-delgadas o gruesa-delgada que tendrán anchos promedio dentro de la norma y serán difíciles de detectar en este nivel. Ejemplo de esto es el par *fi*.

Las letras itálicas son de los mas difícil de tratar, dado que los bloques de palabras no se pueden segmentar con rectas paralelas al eje *y*. Para tratar con este tipo de texto deberemos obtener una estadística de la inclinación para determinar el ángulo al que intentaremos la segmentación.

El producto de esta etapa, será una estructura de lista ligada en donde cada nodo es un espacio en blanco ó una lista de alternativas de segmentación colocando en primer término la alternativa mas probable y en orden descendente las opciones de segmentación posteriores. Esta estructura es útil cuando procedemos a clasificar, cada alternativa en la sublista es calificada con alguna probabilidad de acuerdo al grado de certeza de la clasificación, el clasificador genera una nueva lista en donde cada elemento es una lista de candidatos con un orden dado ahora por su calificación en la clasificación.

En la etapa de postprocesamiento se utilizaría la lista de candidatos para tratar de localizar ahora palabras válidas en el diccionario del sistema con diversas permutaciones de las letras que forman las palabras. Este proceso toma gran cantidad de tiempo si existen muchos alternativas para cada palabra; por ejemplo si una palabra de 8 letras tiene tres alternativas en cada letra, tendríamos que hacer 3^8 consultas al diccionario (19,683 consultas).

La consulta jerárquica descrita no está implementada, la búsqueda en el diccionario será explicada en el capítulo correspondiente.

3

Extracción de características

Aún cuando en el orden natural (en el esquema del reconocimiento de patrones) la extracción de características (EC) se realiza antes que la clasificación, su importancia en el problema es clara cuando se han discutido los clasificadores; sin embargo, discutiremos con argumentos geométricos las ideas principales en este capítulo.

Los patrones (arreglos numéricos que representan a los objetos) contienen información tanto endógena como exógena. La información endógena está referida a las configuraciones que pueden tomar las n-adas de números que representan a cada patrón; la distancia que tengan entre sí los patrones y su distribución en el espacio de búsqueda. La información exógena es aquella que tiene el patrón referido a su interrelación con otros objetos del mundo físico; los patrones son representaciones de objetos reales y nos interesa replicar lo que realiza una persona al clasificarlos, la persona que realiza la clasificación es la fuente de información exógena. La información exógena tiene dos vertientes, la etiquetación de los puntos del conjunto de entrenamiento (capítulo 4) y la extracción supervisada de las características de los objetos.

La EC juega dos papeles dentro de la clasificación, el primero consiste en un reducción dimensional del espacio de búsqueda y el segundo en una corrección geométrica que produce cúmulos de objetos de la misma clase. La reducción dimensional se traduce en una reducción del número

de cálculos necesarios para medir distancias entre objetos; la corrección geométrica o compactación significa la generación de algoritmos de clasificación mas simples y de comportamiento mas estable.

3.1 Definiciones

Los objetos son puntos x del espacio de búsqueda R^d. Las características son funciones $f:R^d \rightarrow R$, que pueden estar dadas analíticamente o especificadas proceduralmente mediante algún algoritmo. La extracción de características equivale al mapeo

(3-1) $\quad F:R^d \rightarrow R^{d'}, \quad F(x)=(f_1(x),...,f_{d'}(x)) \quad , \; d' \leq d$

con $f_i(x)$ la i-ésima característica extraída del vector x.
En el caso de reducción dimensional, d' es estrictamente menor que d; en el caso de compactación d'=d.
Algunos autores llaman a las funciones $f_i(x)$ *características de alto nivel*, y a las entradas del vector x *características de bajo nivel*.
Esta distinción es debida a que en el momento de la adquisición de los datos los objetos son arreglos numéricos de longitud variable que depende de la resolución del dispositivo de entrada, el tamaño del objeto etc. La idea entonces es que las características de bajo nivel son normalizaciones de los objetos y las características de alto nivel son propiamente las características de los objetos.

3.2 Normalización

Al realizar la digitalización de un objeto real obtendremos un arreglo numérico que refleja "una vista" del objeto desde el dispositivo de entrada. La longitud del arreglo numérico y su estructura interna dependen del dispositivo tanto como de la posición del objeto ante el digitalizador. (fig. 3.1)
La normalización consiste en reescribir el arreglo numérico de tal forma que 1) Los objetos tengan la misma orientación y 2) El arreglo numérico tenga la misma longitud y estructura interna en todos los objetos.

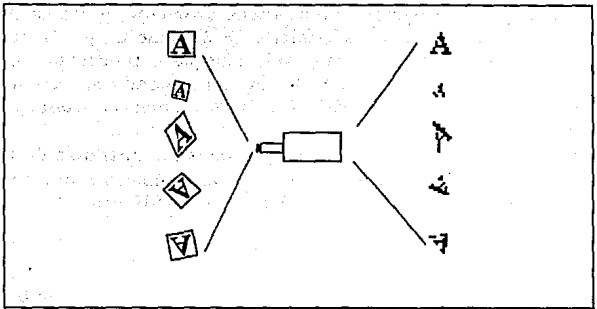


Figura 1
Variedad de posiciones frente al digitalizador

El punto 1 tiene sentido cuando estamos trabajando con objetos cuya digitalización depende de su orientación con respecto al dispositivo de entrada y queremos construir un clasificador que utilice "pocos" ejemplos, i.e. con los objetos dados en forma canónica. El punto 2 equivale a "ver" a todos los objetos con la misma resolución.

el caso de estudio

Las letras obtenidas en la etapa de segmentación son arreglos numéricos con una estructura que es reflejo del método de digitización. En este caso, las letras (o mas propiamente los bloques candidatos a letras) son matrices de pixels cuyo tamaño depende de la resolución del dispositivo de entrada y el tamaño de la letra. Dos letras iguales (de la misma clase) estarán representadas por matrices de dimensiones (posiblemente) diferentes. Describimos como fueron normalizadas las letras:

Los caracteres presegmentados se colocan en una rejilla de $m \times n$ cuadros. Las dimensiones de la rejilla se obtienen del ancho y alto promedio de los caracteres muestra mediante la relación

(3-2) $$m = \frac{\bar{a}}{\bar{a}} n$$

con $\overline{a}$ el alto promedio y $\overline{a}$ el ancho promedio de los caracteres de la muestra. La ecuación (3-2) tiene un parámetro libre n; es decir, para cada selección de n tendremos una m. Este parámetro es de importancia cuando discutamos el tratamiento multiescala (capítulo 5), por lo pronto supongamos que es fijo.
Con los datos de la rejilla S_{ij} llenamos las entradas de un vector de dimensión mn en donde cada número representa el promedio de gris en el cuadro correspondiente

(3-3) $\quad v_{i+(j-1)m} = S_{ij} \quad i=1,..,m \; j=1,..,n$

La figura 3.2, presenta un conjunto de letras a) como se ven en el papel, b) al digitalizarlas, c) al normalizarlas.

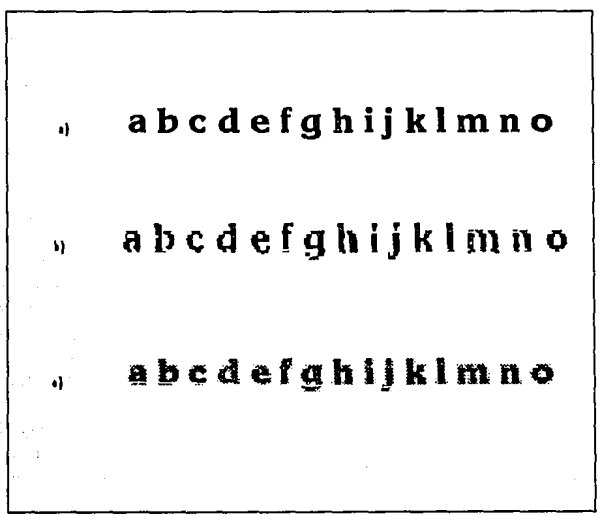


figura 3.2
Normalización de las letras

3.3 Extracción de características

La construcción de las características $f_i(x)$ puede ser dada automáticamente o ad hoc; esto quiere decir, que pueden ser extraídas de los datos mediante consideraciones estadísticas o por el contrario, pueden ser diseñadas por una persona que tenga conocimiento del problema.

El objetivo de la EC es producir un espacio de búsqueda de dimensión tratable (como se verá en los capítulos siguientes la complejidad computacional de un clasificador es función del número de prototipos y de la dimensión del espacio de búsqueda) y de características de separabilidad adecuadas. Si tomaramos directamente los datos normalizados para un problema de reconocimiento de voz, tendríamos que trabajar con vectores de dimensión muy alta (miles de entradas), con el correspondiente costo al calcular distancias; sin embargo, si construimos una transformación que preservando diferencias entre clases disminuye la dimensión de los vectores, ganamos en velocidad de procesamiento sin perder capacidad de diferenciación.

extracción de características ad hoc

Si preguntamos a una persona como es que diferencía una letra de otra, su respuesta va a involucrar propiedades de alto nivel; por ejemplo nos puede decir que la diferencia entre la m y la n estriba en que la primera "tiene dos arcos cóncavos hacia abajo pegados en el centro" y la segunda "tiene un solo arco cóncavo hacia abajo". Otra versión puede ser que la primera "consta de tres trazos verticales conexos en la parte superior" y la segunda "tiene dos trazos verticales conexos en la parte superior". Este proceso (de preguntar como diferenciar una letra de otra) lo podemos extender hasta que agotemos todo el abecedario. Al final, tendremos una lista de propiedades de alto nivel que podemos medir a cada letra para encontrar cuales se cumplen. Si podemos imitar este proceso mediante la aplicación de una serie de funciones a los arreglos numéricos segmentados de una escena, entonces estamos construyendo un vector de características ad hoc para el problema de reconocimiento de letras.

Para ejemplificar esto, pensemos en construir un mecanis-

mo para medir trazos verticales, horizontales y diagonales a 45° y -45°.

Una manera de medir estas características, consiste en utilizar un autómata celular como el definido en el capítulo de preprocesamiento. Las reglas de evolución tendrán una notación definida como sigue

Sea

(3-4)
$$T=\begin{pmatrix} a_{11} a_{21} a_{31} \\ a_{12} a_{22} a_{32} \\ a_{13} a_{23} a_{33} \end{pmatrix}$$

con a_{ij} un símbolo en el conjunto {1,0,*}, donde * es un don't care (puede ser 0 o 1). Una regla de evolución del autómata estaría dada por la terna (T,s,n), con s,n en el conjunto {0,1,*,#} y que significa "si para cada pixel y sus vecinos se cumple con el templete, entonces cambia el centro a s, si no se cumple, cambia el centro a n", el símbolo * significa dejar el centro sin cambio y # invertir el color del centro.

Si tomamos por ejemplo el templete

(3-5)
$$T=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

y la terna (T,1,0), eliminaremos todas las agrupaciones de puntos que no sean rectas a -45°.

Construimos el vector de características contando el porcentaje de pixels que quedan encendidos al iterar 2 veces las reglas (T0,1,0) , (T1,1,0), (T2,1,0), (T3,1,0) con las matrices definidas

(3-6)
$$T0=\begin{pmatrix} 1 & * & * \\ * & 1 & * \\ * & * & 1 \end{pmatrix}$$

(3-7)
$$T1=\begin{pmatrix} * & * & * \\ 1 & 1 & 1 \\ * & * & * \end{pmatrix}$$

(3-8) $$T2=\begin{pmatrix} * & * & 1 \\ * & 1 & * \\ 1 & * & * \end{pmatrix}$$

(3-9) $$T3=\begin{pmatrix} * & 1 & * \\ * & 1 & * \\ * & 1 & * \end{pmatrix}$$

Los vectores de características tendrán entonces dimensión 4. Lo que estaríamos midiendo sería el porcentaje de pixels alineados en las direcciones dadas para cada caracter.

La figura 5.3 muestra la aplicación de estas reglas a una serie de letras, notamos como para cada una de ellas la cantidad de pixels obtenidos al final de las iteraciones depende de si había trazos en la dirección dada.

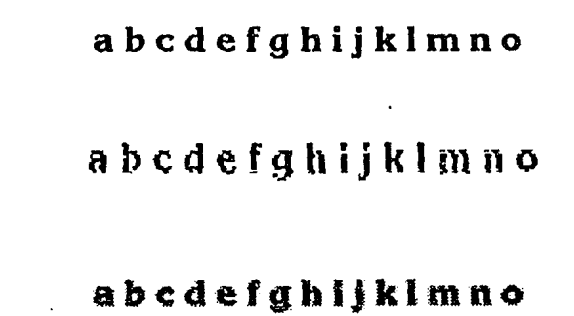


Figura 5.3
Selección de trazos en direcciones enteras

Dado que la cantidad de pixels que sobreviven a la aplicación de la regla es función tanto del grosor de las letras como de la existencia de trazos en las direcciones dadas, eliminamos la dependencia del grosor mediante una división por la cantidad de pixels que originalmente tenía la letra. Los vectores de características tendrían entonces en cada entrada el cociente P(c)/P(f(c)), donde P mide la cantidad de pixels encendidos y f es la aplicación del autómata celular y c el caracter.

extracción automática de características

Si tenemos un problema general de reconocimiento de patrones, no siempre es posible encontrar un extractor ad-hoc que nos produzca elementos típicos o bien puede ser un proceso muy caro computacionalmente (si por ejemplo requiere de la aplicación de transformaciones en el espacio de frecuencias). Es por esto que se ha establecido una metodología de extracción de características cuya filosofía es tomar los vectores de características de bajo nivel (o vectores normalizados) y obtener de ellos las coordenadas mas representativas i.e. aquellas que producen una mejor separación de los datos. Una manera de lograr esto es aplicar una descomposición en componentes principales a la muestra de datos, lo cual equivale a encontrar los eigenvalores de la matriz de covarianzas y hacer una proyección del espacio de búsqueda en las coordenadas mas significativas.

3.4 El problema geométrico

La normalización permite dar un tratamiento geométrico a los patrones. El método de normalización que se emplea en cada problema particular es diseñado para que refleje similitudes y diferencias entre los patrones; depende del conocimiento a priori que se tenga del problema y de las representaciones que entrega el dispositivo de entrada.

Desde el punto de vista de la información contenida en los patrones (los objetos normalizados) observamos que la extracción de características es una reducción de la cantidad de información y un aumento de su calidad. Esto está en concordancia con el planteamiento hecho por K.S. Fu en

[10], en donde se establece un criterio para mediar entre la cantidad de información, su calidad y el poder predictivo de un clasificador.

Cuando el problema de reconocimiento de patrones se circuscribe al paradigma geométrico, egún nuestro particular punto de vista, la extracción de características puede ser embebida en el mismo proceso de clasificación, i.e. podemos diseñar un clasificador que realiza la EC como una etapa interna del algoritmo. El argumento que sustenta esta afirmación consiste en considerar los clasificadores basados en redes neuronales artificiales, tomando el ejemplo mas sencillo (el perceptrón de n capas), podemos pensar en las capas ocultas como extractores de caracterísstcas y en la última capa como un clasificador lineal de fisher sobre los vectores procesados por la capas anteriores. Una discusión en este sentido puede ser encontrada en Pao [3]. Por otro lado cuando el problema es atacado con herramientas sintácticas, la extracción de características es el paso que nos permite plantear el problema. Dada la naturaleza no numérica del problema sintáctico, no existe forma de incluir en el clasificador a la extracción de características.

Calidad de la información

Existen razones pragmáticas y formales para realizar extracción de características de los objetos. De entre las razones pragmáticas podemos citar la reducción del número de cálculos necesarios para clasificar a los objetos; es evidente que si los objetos tiene menos componentes las comparaciones entre objetos llevan menos tiempo. Esta reducción del número de cálculos no debe estar en detrimento de la capacidad predictiva en el sistema de reconocimiento de patrones; es por ello que debemos eliminar solo la información redundante. En cuanto a las razones formales, estas tienen relevancia cuando consideramos la geometría del espacio de representación; es decir, cuando diseñamos un clasificador. La extracción de características (e.c.) propondrá espacios de representación que tienen propiedades geométricas que permiten, nuevamente, construir clasificadores mas simples.

La discusión anterior trae como consecuencia natural una pregunta acerca de la calidad de la información que se obtiene al digitizar un objeto. Es claro que para propósitos de clasificación (el fin último del reconocimiento de patrones) por cada objeto nos basta solo un número que identifique la clase; esta es una tarea de la etapa de clasificación ¿Que es entonces la extracción de características? la única respuesta posible es: Es una manera de facilitar la tarea del clasificador i.e. un procedimiento por medio del cual eliminamos información redundante con el fin de construir clasificadores mas simples.

3.5 Información exógena y endógena

La afirmación contundente del último párrafo de la sección anterior tiene matices que es importante discutir. Lo primero es ¿Como determinar que información es redundante y cual es imprecindible?, ¿Como establecer un nuevo espacio de representación cuya geometría esté acorde con el clasificador que nos proponemos utilizar?. En general ¿Cómo realizamos la resención de la información?

Encontrar la transformación que nos lleve a condensar la información de los objetos tiene dos vías a saber; una automática, en donde se construye un sistema que trabaja en conjunción con el clasificador y se itera hasta encontrar una transformación que haga que el clasificador trabaje eficazmente. La segunda en donde el diseñador del sistema utiliza información a priori para construir el e.c.

Para el primer caso diremos que estamos utilizando información endógena y en el segundo información exógena. Si utilizamos el primer método estaremos haciendo extracción automática de características, en el segundo propondremos transformaciones ad hoc para cada problema.

¿Cual será el método mas adecuado para una aplicación particular? En el primer caso, es obvio que no podemos enriquecer (inyectar información) al conjunto de objetos; únicamente podremos eliminar información redundante. En el segundo caso, el conocimiento a priori del problema que tiene el diseñador es una fuente de información, de donde concluímos que sí estamos en posibilidades de enriquecer la información sobre el conjunto de objetos.

3.6 Metaclases y clusters

Un clasificador no supervisado, que no discutiremos en el capítulo siguiente, trabaja de manera estrecha con la etapa de extracción de características, discutiremos algunas generalidades.

Supongamos que se ha definido un conjunto de características que se medirán al conjunto de entrenamiento, (podemos pensar en el ejemplo concreto de las letras). De todo el conjunto de ejemplos existirán algunos elementos que, independientemente de su clase, compartirán características (por ejemplo tendrán valores positivos dentro de algún rango). La construcción de un clasificador no supervisado consiste en caracterizar las intersecciones de los conjuntos de características. Discutimos un ejemplo ilustrativo:

Supongamos que de alguna manera podemos medir las características geométricas y topológicas de las letras (p/ej con transformaciones como la discutida en la sección 3.2). Supongamos que por cada letra del alfabeto podemos saber si tiene ciclos y cuantos, cuantas líneas horizontales la forman, cuantas líneas verticales etc. Formamos entonces una metaclase que constará de todos los caracteres que comparten características semejantes. De esta forma la metaclase de las letras con ciclos será [o,a,e,b,q,p,d,g,], la metaclase de las letras con trazo vertical será [b,d,f,g,i,j,k,l,p,q,t] y así con las demás características.

Diremos que el sistema de extracción es completo cuando cada clase esté caracterizada por la intersección de algún conjunto de metaclases.

Este clasificador es solo uno de muchos que se pueden construir con el mismo espíritu; a saber, haciendo interactuar el clasificador con el extractor de características.

3.7 Conclusiones

La extracción de características corresponde a una descripción de alto nivel de los objetos a clasificar. Si el problema es geométrico y supervisado, la EC constituye un medio para construir espacios de búsqueda mas tratables. Si el problema es geométrico y no supervisado, la EC es un mecanismo que permite encontrar metaclases o clusters en

el espacio de búsqueda. Si el problema es sintáctico, el extractor de características es el constructor de primitivas del lenguaje formal en el que se trabajará.

4

Clasificación

4.1 Definición del problema

El paradigma geométrico de reconocimiento de patrones establece como criterio de semejanza entre objetos la distancia que exista entre sus representaciones numéricas, el grado de semejanza entre objetos será inversamente proporcional a su distancia. Hablaremos de los objetos como puntos de un espacio métrico d-dimensional. Este espacio será denominado indistintamente espacio de búsqueda; los objetos serán vectores o patrones.

Un clasificador particiona al espacio de búsqueda en regiones disjuntas que además lo cubren. A cada una de estas regiones se les asigna una etiqueta que indica una clase o indica rechazo. A un objeto desconocido se le asignará una etiqueta de clase (o se rechazará) de acuerdo a la región en la que caiga fig. 1. Las regiones que define el clasificador serán conocidas como regiones de decisión.

El entrenamiento de un clasificador es precísamente la etapa en la que se determinan la regiones de decisión. Este entrenamiento se realiza obteniendo estadísticas de un conjunto de ejemplos etiquetados con la clase a la que pertenecen; este conjunto es llamado conjunto de entrenamiento.

Los clasificadores serán entonces categorizados por el tipo de regiones de decisión que pueden generar y por el tipo de entrenamiento que se les dá. El tipo de entrenamiento puede ser supervisado, que consiste en etiquetar adecuadamente cada elemento de la muestra conocida; o puede ser no supervisado, en donde se analizan los cúmulos que

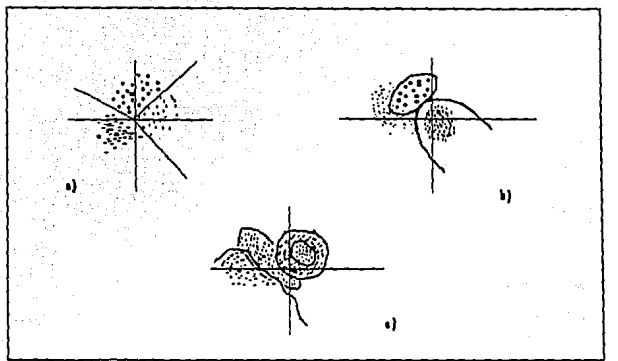


Figura 1
Partición del espacio de búsqueda
Diferentes regiones de partición: a) lineal, b) Cuadrática, c) General

forman los objetos del conjunto de entrenamiento para decidir la partición que se hará del espacio de búsqueda; dicho de otra manera, cuando desconocemos a priori la la identidad de los distintos patrones del conjunto de entrenamiento estamos en el caso no supervisado.
Para el caso particular de reconocimiento óptico de caracteres que nos ocupa, ensayamos dos tipos de clasificadores; uno orientado a la búsqueda de parámetros de funciones discriminantes y un método no paramétrico que utiliza todo el conjunto de entrenamiento.

4.2 Clasificadores hiperesféricos

El vecino mas cercano

Consideremos el conjunto entrenamiento $\{E_i,w_i\}$ con E_i el i-ésimo vector del conjunto y w_i su clase asociada, y un vector desconocido x al que queremos clasificar. Para determinar la clase a la que pertenece x un procedimiento plausible consiste en encontrar dentro del conjunto $\{E_i,w_i\}$ el vector mas cercano E_s bajo la distancia dada y asociarle a x la etiqueta w_s , es decir

w_s es la clase de x sii $||E_s - x|| < ||E_i - x||$ para $i \neq s$ (4-1)

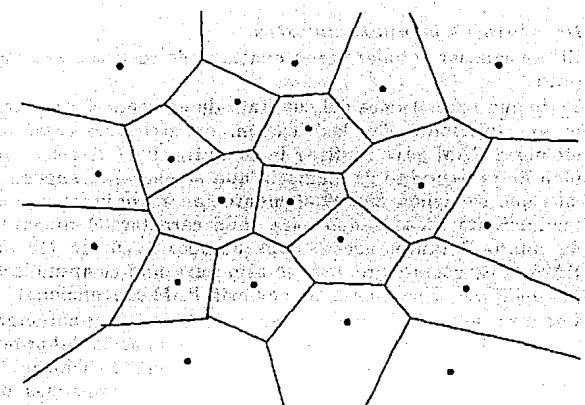


Figura 2

Diagrama de Voronoi en 2 dimensiones

El criterio del vecino mas cercano define una región poligonal en la que cada arista es la frontera que separa a dos puntos. Cada polígono en la figura es la región de los puntos mas cercanos al punto del centro.

Este procedimiento define para cada elemento del conjunto de entrenamiento un hiperpoliedro de Voronoi. El conjunto de todos los hiperpoliedros es llamado un diagrama de Voronoi. La figura 2 muestra un diagrama de voronoi en dos dimensiones.

Este procedimiento es razonable, dada la suposición a priori de que los patrones son semejantes cuando son cercanos bajo la distancia dada, pero es solo un aproximación relajada a la solución del problema. La principal desventaja consiste en calcular N distancias, N el número de elementos del conjunto de entrenamiento, que en general son costosas cuando la dimensión del espacio de búsqueda es alto. Existen técnicas para disminuir el número total de cálculos para localizar el vecino mas cercano, discutiremos algunas en el capítulo 5.

Analizamos algunas características del clasificador del vecino mas cercano.

Desventajas computacionales

El algorimtmo requiere gran cantidad de recursos de computo.

Dado que necesitamos del conjunto de entrenamiento para el procedimiento de clasificación, es necesario reservar memoria RAM para realizar la consulta. Para darnos una idea de la cantidad de memoria que se necesita, supongamos que se tienen 30,000 elementos en el conjunto de entrenamiento. Reservando para cada caracter 64 números de punto flotante, necesitamos $30{,}000 \times 64 \times 8 = 15$ Mb en RAM. Este costo es realmente alto para una computadora personal que tiene .5 Mb de memoria RAM convencional.

Por otro lado, el tiempo de clasificación para un caracter, descontado el tiempo de preprocesamiento, sería del orden de 64x30,000 sumas + 64x30,000 multiplicaciones + 30,000 raíces cuadradas = 3,870,000 de operaciones de punto flotante (suponiendo que se utiliza la distancia euclideana). En una máquina que puede realizar .1 Mflops (100,000 operaciones de punto flotante por segundo) representa 38.7 segundos; para una página de texto tendríamos alrededor de 1500 caracteres, es decir: ¡16 horas!

Este es un costo que no se puede obviar. De no poder mejorar esta marca de tiempo, solo podemos pensar en la construcción de una máquina de propósito específico que realice clasificaciones en paralelo; de otra manera, este algoritmo representa solo una posibilidad teórica de resolución.

Desventajas de clasificación

Desde el punto de vista del poder predictivo de este clasificador, podemos observar algunas desventajas cuando tenemos condiciones no ideales para la construcción del conjunto de entrenamiento.

1)- El conjunto de entrenamiento no es representativo

Cuando los caracteres que forman al conjunto de entrenamiento no son representativos de los que vamos a encontrar en la aplicación, bien sea que tengan una calidad diferente o hayan sido adquiridos con un dispositivo de entrada distinto al utilizado en la aplicación, tendremos un alto número de

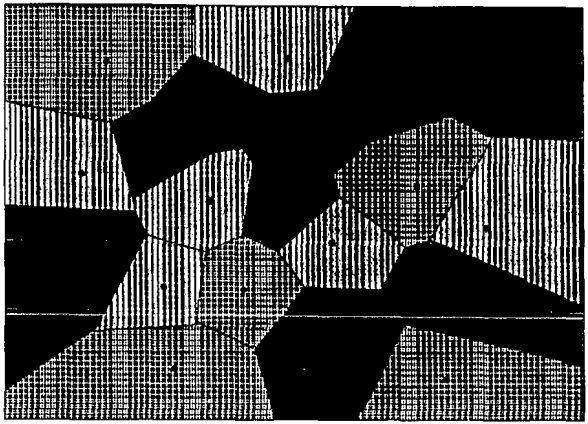


Figura 3

Una muestra no representativa

Cuando la muestra del conjunto de entrenamiento no es representativa del problema, los caracteres que llegan son mal clasificados. Todas las fronteras de los polígonos y en particular las intersecciones son regiones de indefinición

falsos positivos i.e. caracteres que por ser defectuosos caen en elementos de partición equivocados (fig. 3).

2)- *El patrón que se quiere clasificar tiene ruido*
Este caso es semejante al anterior; cuando el patrón que se quiere clasificar difiere sustancialmente de los que tienen su clase en el conjunto de entrenamiento, tendremos también una clasificación falsa.

3)- *Existe mas de un vecino cercano*
Este es un caso crítico para este tipo de clasificadores, corresponde a localizar un patrón precisamente en la frontera de los hiperpoliedros de Voronoi de varios puntos del conjunto de entrenamiento (fig 3 *). El algoritmo de k-vecinos que se verá mas adelante puede solucionar conflictos de esta naturaleza.

4)- *La calidad de la clasificación no está controlada, no hay un cálculo explícito del riesgo de rechazo.*
Cuando un comprador potencial de un sistema de OCR solicita las características del sistema, busca que el riesgo de clasificación cruzada sea mínimo; es decir, un error de rechazo

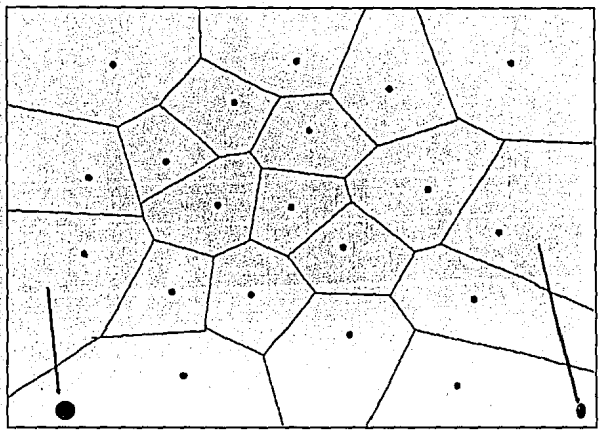


Figura 4

Un vecino no cercano

Cuando existe mucha separación entre muestras de caracteres (ejemplos aislados), un caracter será clasificado en la clase mas cercana; aún cuando esta se encuentre lejos de él.

es mas deseable que un error de clasificación cruzada: el error de rechazo es detectable y el de clasificación cruzada es mas difícil de tratar: marcados los caracteres rechazados un sistema automático o manual puede corregir los errores, si hay clasificación cruzada es mas difícil corregirlos, habría que emplear un diccionario muy grande y varias consultas a él.

Dado que no hay regiones de rechazo en este clasificador, construido tal como se describe arriba, cualquier caracter es clasificado al vecino mas cercano; aún cuando este se encuentre realmente "lejos" de alguna región de clasificación. (Fig. 4)

Ventajas computacionales

1)- Es un algoritmo no paramétrico

Cuando se trata de poner a punto un clasificador, la determinación de parámetros consume la mayor cantidad de tiempo. Las características que se extraen de los patrones son propuestas por personas familiarizadas con el problema; pero una característica particular de alto nivel no siempre

es adecuada para un problema geométrico. Probar la calidad de la extracción de características exigiría poner a punto (parametrizar) el clasificador. La calidad de la clasificación siempre estará acotada por abajo (al menos se puede esperar *esta calidad* de clasificación) por un clasificador del vecino mas cercano.

Ventajas de clasificación

Este clasificador es equivalente a una red neuronal artificial alimentada hacia adelante (feedforward network).

De acuerdo a diferentes reportes[], este clasificador tiene un poder predictivo semejante a las redes neuronales alimentadas hacia adelante. Esto da una idea para la construcción de algoritmos paralelos para acelerar el tiempo de clasificación.

1)- En problemas de dimensión baja funciona adecuadamente

Si el número de entradas de los vectores con los que se trabaja es pequeño, la cantidad de operaciones disminuye. Si adicionalmente tenemos una distancia no euclideana (como la distancia de Hamming) cuyo cálculo no sea costoso, el clasificador del vecino mas cercano es adecuado.

3)- Funciona adecuadamente en problemas no linealmente separables

Este clasificador es recomendable sobre el clasificador lineal de fisher cuando el espacio de búsqueda no es linealmente separable (ver el siguiente apartado).

Los K vecinos mas cercanos.

Para solucionar problemas de empate, o cuando se quiere garantizar un mínimo de seguridad de rechazo, se puede generalizar la idea del clasificador del vecino mas cercano haciendo una votación:

1)- Se localizan los k-vecinos mas cercanos (k-vc) al vector desconocido x.

2)- Se determina por una votación la clase a la que pertenece x, y será aquella que tenga mayoría.

3)- En caso de empate, se rechaza al caracter.

4)- Si no se cumple con un mínimo de vecinos con la misma clase, se rechaza al caracter.

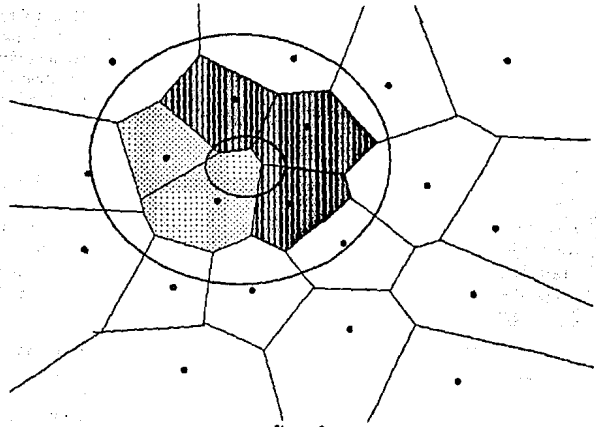


Figura 5
Diagrama de Voronoi para los k-vc
Cualquier patrón que caiga en el círculo pequeño será clasificado como de la clase 2 (gris oscuro).

La figura 5 muestra los casos de aceptación.

Adicionalmente, el clasificador k-vc se puede generalizar con una quinta regla que involucra la distancia a la que se localizan los vecinos, tomando un voto pesado por la distancia.

Obviando la complejidad computacional del clasificador k-vc, el principal problema desde el punto de vista del poder predictivo es la determinación del número óptimo de vecinos a considerar. Este número se determina ad-hoc en las aplicaciones, haciendo experimentos con un conjunto de prueba tratando de minimizar el número de rechazos y de clasificaciones cruzadas.

Para el caso particular de reconocimiento óptico de caracteres, una representación binaria minimiza la cantidad total de memoria utilizada para representar el conjunto de entrenamiento, y utilizando la distancia de hamming para comparar patrones el cálculo de la distancia se hace mas sencillo.

Para el mismo ejemplo de 30,000 caracteres en el conjunto de entrenamiento, con 64 entradas cada vector, la cantidad de memoria requerida es: 30,000x64/8= 234Kb, cantidad razonable en una computadora personal; 64 entradas binarias son representadas en 8 bytes. Explicamos el cálculo de la distancia:

Sea d el número de entradas de los vectores binarios, $\{v_i\}$ el conjunto de entrenamiento

1)- Tomamos v_i y x_i los dos vectores para comparar, hacemos

$s_i = v_i$ XOR x_i ,i=1,..., $\frac{d}{8}$

En s_i está almacenado el vector que contiene las diferencias de v_i y x_i.

2)- Contamos el número de bits encendidos: esa es la distancia de hamming.

Normalmente necesitaríamos contar el número de bits encendidos en el vector s_i, que implicaría hacer d máscaras, 2d corrimientos y d incrementos en una variable que cuente el número de bits. Esto implica hacer 4d+d/8 operacio-

0	00000000	0
1	00000001	1
2	00000010	1
3	00000011	2
4	00000100	1
.	.	.
.	.	.
.	.	.
254	11111110	7
255	11111111	8

figura 6

Una tabla para sacar el número de bits

Cada entrada de la tabla contiene el número de 1's que tiene el índice de esa entrada. De esta manera, contar el número de bits en un byte es hacer una consulta a la tabla.

nes en total. Proponemos un procedimiento alternativo para contar el número de bits en el vector s_i:

> Construimos una tabla de tamaño 256, en donde almacenamos el número de bits que tiene el índice de cada entrada, por ejemplo en T[128] colocamos un 1 (fig. 6). De esta manera contar el número de bits en s_i implica hacer d/8 sumas y d/8 consultas, de donde tendríamos que el número total de operaciones para obtener la distancia de hamming sería d/4.

Para el caso particular que nos ocupa, el localizar el vecino mas cercano tomaría 30,000x64/4=48,000 operaciones elementales. En una computadora personal esto toma del orden de .25 segundos, se reduce el número de cálculos por un factor de 150 con respecto de la representación en números de punto flotante; sin embargo, el tiempo total para traducir una hoja de texto es muy alto; del orden de 50 minutos. En el capítulo de tratamiento multiescala, daremos un remedial para esta situación, que consiste en acotar la búsqueda eliminando posibilidades: no tenemos que buscar en toda la base de datos.

Generalización, el problema hiperesférico

Discutimos es esta parte una generalización del clasificador k-vc, esta no fué instrumentada en el trabajo, queda como una propuesta para extensiones a futuro.

La desventaja principal de los métodos tipo vecino mas cercano es que se tiene que utilizar toda la base de datos para hacer predicciones. Una manera de remediar esto es reducir el número de elementos en el conjunto de entrenamiento a condición de que no se pierda poder predictivo, i.e. que al menos se tenga la cota de reconocimiento del clasificador k-vc. Discutimos una opción en los párrafos siguientes.

Búsqueda de cúmulos homogéneos.

Cuando en el conjunto de entrenamiento existen cúmulos de objetos de la misma clase, cualquier patrón que caiga en estas regiones será clasificado en consecuencia. La idea es buscar este tipo de cúmulos haciendo iteraciones hasta localizar un cúmulo maximal encerrado por una región de geometría específica. Una de las regiones mas tratables desde el punto de vista computacional es una hiperesfera localizada en el centroide del cúmulo, con radio tal que cu-

bra solo elementos de la misma clase. Este proceso se repite en cada cúmulo homogéneo. Cuando se tenga el caso de regiones de puntos aislados, el centro de la hiperesfera coincide con el punto y radio es la mitad de la distancia al vecino mas cercano (que en este caso es un objeto de otra clase).

La clasificación consiste en localizar la hiperesfera en la que se encuentra el objeto desconocido. Si un objeto no cae dentro de ninguna hiperesfera es rechazado.

En el peor de los casos, cuando no se pueden construir cúmulos homogéneos, este clasificador coincide con el clasificador del vecino mas cercano (desde el punto de vista del poder predictivo); sin embargo, hay nuevas regiones de indefinición o de rechazo (los huecos que quedan al tratar de cubrir el espacio de búsqueda). En el caso general, se espera que el uso de este clasificador disminuya el número de cálculos necesarios y con ello el tiempo total de clasificación.

4.3 Funciones discriminantes

Si conociéramos la distribución de probabilidad del conjunto muestra, construir el clasificador significaría utilizar la regla de bayes para convertir la probabilidad a priori en una probabilidad a posteriori. No podemos conocer la distribución de probabilidad, solo podemos hacer suposiciones acerca de ella. Suponiendo una distribución de probabilidad teórica, podemos resolver un conjunto limitado de problemas. Discutiremos en este apartado la solución del problema de clasificación cuando la función de distribución es multinormal.

Funciones discriminantes lineales

Clasificador lineal de Fisher.

Si conociéramos la distribución de probabilidad a priori del conjunto de entrenamiento para cada una de las clases, podríamos convertirla en una probabilidad a posteriori utilizando la regla de Bayes:

$$P(w_i|\,\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\,w_i)\,P(w_i)}{p(\boldsymbol{x})} \qquad (4\text{-}1)$$

con

(4-2) $$p(x) = \sum_i p(x|\, w_i)\ ,$$

$P(w_i)$ la probabilidad a priori de ocurrencia en cada clase, $p(x|\, w_i)$ la probabilidad condicional del vector x y w_i las clases de las que se compone el problema.
Consideremos la distribución de probabilidad $p(x)$ de acuerdo al modelo gausiano (multinormal),

(4-3) $$p(x) = (2\pi)^{-\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right]$$

con x un vector d dimensional, media μ y matriz de covariancias Σ. Supondremos que para cada i $p(x|\, w_i)$ se distribuye de acuerdo a (4-4). La diferencia entre clases está dada tanto por las medias μ_i como por las matrices de covariancia Σ_i, que son los parámetros del clasificador que discutiremos.
Definimos una función discriminante para la i-ésima clase como

(4-4) $$g_i(x) = P(w_i|x)$$

Clasificaremos a x como correspondiente a la clase para la cual la probabilidad a posteriori sea máxima; es decir:
x pertenece a w_i sii $g_i(x) > g_j(x)$ *para* $i \neq j$
De acuerdo a este criterio, cualquier función monótonamente creciente sobre una función discriminante es también una función discriminante.
Supongamos que las probabilidades a priori son iguales para todas las clases[1], entonces (4-5), en cuanto función de decisión puede verse como $g_i(x) = p(x|\, w_i)$ de (4-5) y (4-2), dado que el factor $p(x)$ es constante para todas las clases, y lo mismo $P(w_i)$.

1.- Esto equivale a decir que no tenemos esa información, todas las clases son equiprobables a priori.

Para eliminar la dependencia funcional exponencial en (4-4), aplicamos la función logaritmo (que es una función monótonamente creciente) a la definición (4-5) y tenemos

(4-5) $$g_i'(x)=\log\{g_i(x)\}$$

(4-6) $$g_i'(x)=-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)-\left(\frac{d}{2}\right)\log(2\pi)-\frac{1}{2}\log|\Sigma_i|$$

que es también una función discriminante.

Caso 1:

La matriz de covarianzas Σ_i es la identidad, $\Sigma_i=\Sigma=I$. El criterio de separabilidad está dado únicamente por la media μ_i.

En este caso, los términos 2 y 3 de la expresión (4-6) a son constantes aditivas independientes de la clase y pueden ser eliminados de la expresión, el producto por la identidad desaparece obteniendo

(4-7) $$g_i(x)=-\frac{1}{2}(x-\mu_i)^T(x-\mu_i)$$

Desarrollando el término cuadrático en (4-7), obtenemos

(4-8) $$g_i(x)=-\frac{1}{2}(xx^T-2\mu_i^Tx+\mu_i^T\mu_i)$$

La función discriminante (4-8) es llamada el detector de correlación. Obsérvese que el tercer término es constante para cada clase y puede ser calculado a priori, el primer término es común a todas las funciones $g_i(x)$, de tal suerte que podemos reescribir (4-7) para obtener

(4-9) $$g_i(x)=w_i^Tx+w_{i0}$$

con

(4-10) $$w_i=\mu_i$$

(4-11)

(4-12) $$w_{i0}=-\frac{1}{2}\mu_i^T\mu_i$$

Podemos interpretar (4-9) como un templete que colocamos sobre el vector desconocido, para obtener correlación máxima. Este clasificador genera regiones de decisión que son hiperplanos en el espacio de búsqueda; buscamos la distancia euclideana del objeto desconocido a la media de cada clase y tomamos la mínima[2]. Podemos ver a (4-7)c como un perceptrón de una capa, con matriz de pesos w_i y elementos de corte w_{i0}.

Caso 2:

La matriz de covarianzas $\Sigma_i=\Sigma=\sigma^2 I$, los componentes de x no están correlacionados (son estadísticamente independientes).

En este caso, de la ecuación ecuación (4-6) podemos eliminar todavía los términos 2 y tres, que son independiente de la clase. Recordemos que $\Sigma^{-1}=\frac{1}{\sigma^2}I$, de donde obtenemos

(4-13) $$g_i(x)=-\frac{1}{2\sigma^2}(x-\mu_i)^T(x-\mu_i)$$

En este caso, el término cuadrático determina una hiperesfera centrada en la media de la clase y radio 2σ. Este es un mejor clasificador que el discriminante lineal; pero se deben realizar mas cálculos para obtener una predicción. Nótese que este clasificador *no es inherentemente cuadrático*; es decir, todavía podemos hacer la misma simplificación que en (4-7)c, actualizando la constante -1/2 de (4-9).

Caso 3:

La matriz de covarianzas es diagonal, con elementos distintos en cada elemento de la diagonal. $\Sigma_i=\Sigma$, σ_{ii} diferente para cada i.

2.- Nótese como este planteamiento es un caso particular de un clasificador hiperesférico, con un solo cúmulo homogéneo por clase, la región de decisión no es una hiperesfera; sino un polígono de voronoi sobre cada media de clase.

Obtendremos una ecuación discriminante inherentemente cuadrática, en la que debemos calcular el inverso de la matriz Σ, que es también una matriz diagonal, con elementos los inversos de las covarianzas dadas.

(4-14) $$g_i(x) = -\frac{1}{2}(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)$$

con $\Sigma = \begin{pmatrix} \sigma_{11}^2 & 0 & 0 \\ 0 & \sigma_{22}^2 & 0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot\cdot & 0 \\ 0 & 0 & \sigma_{dd}^2 \end{pmatrix}$ y $\Sigma^{-1} = \begin{pmatrix} \frac{1}{\sigma_{11}^2} & 0 & 0 \\ 0 & \frac{1}{\sigma_{22}^2} & 0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot\cdot & 0 \\ 0 & 0 & \frac{1}{\sigma_{dd}^2} \end{pmatrix}$

Razonamos que si en una coordenada específica tenemos una varianza pequeña, en la matriz inversa tendremos un número grande e inversamente. Esto indica que aquellas coordenadas que tengan varianza pequeña, son mas importantes (desde el punto de vista de clasificación) que las que tienen una varianza grande. Las regiones de decisión que se forman son hipercuadráticas; la distancia que estamos midiendo es llamada *distancia de Mahalanobis*.

Para el caso general, podemo hacer una diagonalización de la matriz de covarianzas, esto equivale a encontrar una transformación de coordenadas en la que es posible hacer el mismo trabajo de clasificación; es decir, buscamos un espacio de búsqueda isomorfo al espacio dado, pero mas tratable desde el punto de vista computacional. Esta transformación es llamada *descomposición en componentes principales*.

Discusión de clasificador

Ventajas computacionales

El número total de cálculos que se deben realizar en el modo de clasificación no depende del número de elementos del conjunto de entrenamiento; depende únicamente del número total de clases en el espacio de búsqueda. El tiempo de entrenamiento es alto; sim embargo, es mas razonable utilizar un tiempo alto de entrenamiento y poco tiempo en la clasificación.

Ventajas de clasificación
Cuando el problema que se quiere resolver es linealmente separable, tenemos a la mano una herramienta barata en tiempo de clasificación; de hecho, si el problema es linealmente separable, este clasificador es la mejor alternativa, dado que permite incluir un término de probabilidad a priori que sería de utilidad cuando conocemos estadísticas de la frecuencia de ocurrencia de los patrones en una muestra de clasificación dada.

Desventajas de clasificación
Si el problema no es linealmente separable, las predicciones estarán sesgadas. Mas concretamente, el clasificador no sirve para ningún otro caso. Si hubiera dos o mas cúmulos en cada clase, al calcular la media de la clase caeríamos en una región que posiblemente no está relacionada con las regiones de la clase. (fig. 9)

4.4 El caso particular de reconocimiento texto.

Discutiremos en este apartado los resultados obtenidos con la aplicación de los dos clasificadores: k-vc y el clasificador lineal de fisher.

Caso 1: letra monotipo
Restringimos el problema de reconociento de caracteres para que los patrones de entrada tengan únicamente un tipo de letra. Esta restricción está hecha con el ánimo de construir un espacio de búsqueda que sea linealmente separable; sin embargo las regiones se traslapan cuando tenemos libres los siguientes parámetros:

a). Contraste y golpe de la letra
Una letra **negrita** tiene un grosor distinto a una letra normal. Se encontró que el grosor era hasta cierto punto equivalente al parámetro de contraste de digitización. Cuando se adquiere el texto con un scanner, los colores o tonos de gris de la letra en la hoja de papel son convertidos a blanco y negro; el parámetro de contraste del digitizador es precísamente el punto en el cual los tonos de gris se mapean al negro o al blanco; si el contraste es alto, sig-

nifica que estamos cortando en una región cercana al blanco, si el contraste es bajo estamos cortando en una región cercana al negro. Las letras con contraste alto se ven mas gordas y las de contraste bajo mas delgadas, esto es debido a la resolución espacial finita del scanner que en la frontera de la letra localiza "fracciones" de pixel, y las mapea a tonos de gris. (fig. 10)

Geométricamente, este hecho se traduce en la aparición de varios centroides en una misma clase; uno correspondiente (posiblemente) a las letras claras (de bajo contraste) o delgadas y otro (posiblemente) a las letras negritas o de alto contraste.

Este sesgo se puede evitar preprocesando las letras con un filtro de mediana robusto. El parámetro de contraste estará bajo control después de la aplicación del filtro. Este filtro es explicado en el capítulo de extracción de características.

Se observa que el gasto adicional para seguir manteniendo la separabilidad lineal es la aplicación de este filtro.

Para el caso del clasificador k-vc, este es insensible al contraste de las letras. Una de las razones puede ser que se hace un preprocesamiento por defecto al binarizar las letras i.e. cuando pasamos del vector de números de punto flotante a un vector binario, aplicamos un filtraje de mediana robusto a la letra para determinar el elemento de corte.

b)- Mala calidad en el texto fuente

Para el caso del clasificador lineal de fisher, este es un punto neurálgico. Si en el conjunto de entrenamiento hay letras con errores de digitización, el cálculo de la media resulta sesgado; con el consiguiente error al tiempo de clasificación. Si por otro lado, presentamos una letra que tiene errores de digitización tendremos o bien un rechazo o una clasificación cruzada dependiendo de la cantidad de ruido.

Por el contrario, el clasificador de k vecinos es bastante estable con respecto al ruido en las letras.

Cabe hacer notar que el caso interesante del reconocimiento óptico de caracteres es precísamente cuando tenemos

ruido en las letras adquiridas, es decir, cuando trabajamos en condiciones realistas.

resumen de resultados

k-vc

Para el caso de reconocimiento de texto impreso multifuente (multitipo), el clasificador k-vc no requiere de una inmensa base de datos, bastan con alrededor de 50 ejemplos por cada letra en cada tipo. Un hecho interesante es que si se quieren meter mas ejemplos a la base de datos, encontramos que hay repeticiones i.e. un mismo patrón aparece varias veces en la tabla.

Las características para este problema específico son:

Estable con respecto al ruido en la adquisición.

Estable con respecto al factor de contraste

No está limitado a letras monotipo

El tiempo de entrenamiento es el necesario para binarizar a todas las letras de la base datos.

El tiempo de clasificación es de aproximadamente 50*96*16 operaciones elementales mas el tiempo de binarización, que implica aplicación de un filtro de mediana robusto; es decir de alrededor de un segundo por cada letra. Esto es: para una página de texto, tenemos una cota de 25 minutos si la página está llena (1500 caracteres).

La tasa de reconocimiento es de alrededor del 95%

Clasificador lineal de fisher

Inestable con respecto al ruido en la adquisición

Estable con respecto al factor de contraste. A condición de que se haga un preprocesamiento.

El tiempo de entrenamiento es el necesario para calcular 96 medias, de las 96 clase y las correspondientes varianzas.

El número de cálculos es aproximadamente 96*64*2 operaciones de punto flotante; del orden de un cuarto de segundo por cada caracter. Si se cambia de representación y trabajamos con números enteros en lugar de números punto flotante (después de obtenidos los parámetros del conjunto de entrenamiento), el tiempo de clasificación disminuye a 50 centésimas de segundo por cada letra. En la

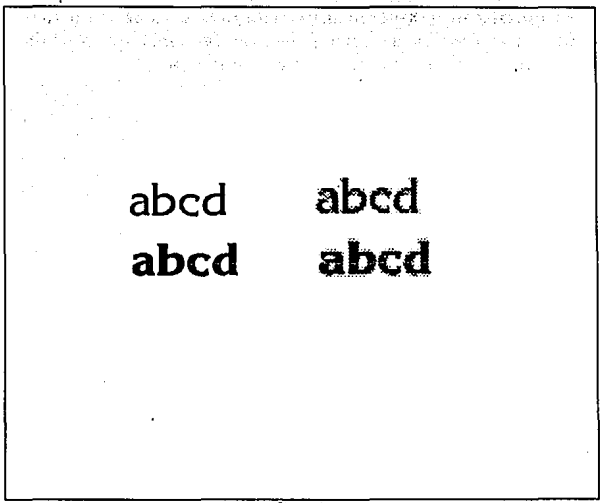


Figura 7
Efecto del contraste en la digitalización
En las fronteras de la letra los pixels se eliminan si el contraste es alto y se agregan si el contraste e alto.

representación con números enteros, la cota de tiempo de reconocimiento para una hoja con 1500 golpes es de un minuto y veinticinco segundos.
La tasa de reconocimiento es de alrededor del 85%
Solo se aplica a letras monotipo.

Conclusiones

La aplicación del clasificador lineal de fisher a un problema de letras monotipo es adecuado. Se pueden obtener mejores tasas de reconocimiento aumentando mas filtros en la etapa de preprocesamiento. Utilizando un buen diccionario al postprocesar el texto, tendríamos un sistema adecuado para aplicaciones muy locales de reconocimiento de caracteres; verbigracia, cuando vamos a digitizar documentos provenientes de una sola máquina de escribir.

El clasificador k-vc es muy poderoso en cuanto a la tasa de reconocimiento, pero presenta la seria desventaja de que tiene un tiempo de cómputo muy alto.

5

Procedimiento multiescala

5.1 El problema

Dado un clasificador que funciona para un problema específico, es nuestra intención disminuir el número total de cálculos necesarios para hacer una clasificación.

El número de cálculos necesarios para una clasificación es función de la dimensión del espacio de búsqueda (d) y del número de prototipos que hay que comparar (k). Si es el caso del clasificador lineal de fisher, k= número de clases; para el clasificador de k vecinos k= número de elementos del conjunto de entrenamiento.

Si podemos disminuir la dimensión del espacio de búsqueda, el número total de cálculos disminuye. Para disminuir la dimensión podemos apelar a la extracción de características; sin embargo, no siempre es posible encontrar funciones ad hoc que provean mecanismos para tal efecto.

Consideremos $\mathbf{R}^d$ el espacio de búsqueda d-dimensional y una proyección

(5-1) $\quad L(x): \mathbf{R}^d \rightarrow \mathbf{R}^{d'} \quad d'<d$

con x' en $\mathbf{R}^{d'}$ calculada coordenada a coordenada

(5-2) $\quad x'_i = \alpha_1 x_{i_1} + \ldots + \alpha_k x_{i_k}, \qquad 1 \leq i \leq d'$

de tal forma que los $\{x_{ij}\}$ forman una partición del conjun-

to de coordenadas $\{x_i\}$. Es decir, cada conjunto $\{x_{ij}\}$ es ajeno y la unión forma el total.
Es fácil ver que si escogemos las $\alpha_i \leq 1$ en (5-2), entonces la norma de cualquier vector x no puede crecer cuando pasamos de R^d a $R^{d'}$.
La norma de $x'=L(x)$ la podemos escribir como

$$(5\text{-}3) \qquad \|x'\| = \sum_i \alpha_i^2 x_i^2 + 2\sum_{i \neq j} \alpha_i x_i \alpha_j x_j$$

aplicando directamente la definición de norma euclideana a (5-2) en cada componente x'_i de x. Nótese que (5-3) es independiente de la selección de la partición.
Observamos adicionalmente de (5-3) que

$$(5\text{-}4) \qquad \sum_i \alpha_i^2 x_i^2 \leq \sum_i x_i^2 = \|x\|$$

para $\alpha_i \leq 1$. Dado que $\|x\| \geq 0$ y $\sum_i \alpha_i^2 x_i^2 \geq 0$ podemos deducir de (5-3) y (5-4) que $\|x'\| \leq \|x\|$.
De esta manera, dos puntos cercanos en el espacio de búsqueda original seguirán siendo al menos tan cercanos en el espacio transformado, lo que no podemos asegurar es que puntos alejados no se colapsen. De hecho, un número infinito de puntos en el espacio original corresponderán a un solo punto del espacio transformado. Nos preguntamos entonces que sucede con la partición que define el clasificador en el espacio original, en donde cada elemento de la partición corresponde a una clase del problema de clasificación.

Supongamos que en el espacio $R^{d'}$ las clases $\{W_c\}=\{w_{c1}, w_{c2}, \ldots w_{cl}\}$ son indistinguibles, i.e. los elementos de esas clases en el conjunto de entrenamiento son clasificados de manera cruzada. Entonces podemos construir un clasificador con las nuevas clases w_c (del mismo tipo que el clasificador original) de tal forma que al clasificar en $R^{d'}$ lo que nos regresa no es la clase a la que corresponde el elemento desconocido; sino en cual subconjunto de cla-

ses debemos buscar en el espacio $\mathbf{R}^d$.
Este procedimiento proporciona un ahorro considerable en el número de cálculos necesarios para clasificar a un vector, dado que el espacio transformado tiene una dimensión menor que el espacio original y la clasificación ahí es mas barata que la clasificación en el espacio original.
Sea $W_p=\{w_1,...,w_p\}$ el conjunto de clases del problema, K_p el conjunto de prototipos que utiliza el clasificador para realizar una predicción, $W_c=\{\omega_1,...,\omega_c\}$ el conjunto de clases que son indistiguibles en $\mathbf{R}^{d'}$ y K_c el conjunto de prototipos asociados. Podemos pensar que cada w_i tiene asociado un conjunto de prototipos $\{k_i\}$ y cada ω_i un conjunto $\{\kappa_i\}$, de tal forma que

$K_p=\{k_1\}\cup\{k_2\}\cup...\cup\{k_p\}$ y

$K_c=\{\kappa_1\}\cup\{\kappa_2\}\cup...\cup\{\kappa_c\}$.

El clasificador C lo podemos parametrizar con la dimensión del espacio de búsqueda, el número de prototipos que utiliza y el vector que deseamos clasificar, de tal forma que C=C(x,K,d). Para realizar una clasificación en el modo multiescala descrito, procedemos a realizar las operaciones

(5-5) ω_i=C(L(x),K_c,d') en el espacio $\mathbf{R}^{d'}$

y posteriormente realizamos

(5-6) w_i=C(x,κ_i,d) en el espacio $\mathbf{R}^d$

El número total de operaciones que hay que realizar es menor que realizar directamente

(5-7) w_i=C(x,K_p,d)

Esto se puede ver si consideramos la complejidad del algoritmo de clasificación.
Sea O(K,d) la complejidad del algoritmo C, con K el número de prototipos utilizados y d la dimensión del vector de características. Para clasificar con (5-7), debemos realizar

$O(K_p,d)$ operaciones; mientras que para clasificar con (5-5), (5-6) debemos realizar $O(K_c,d') + O(\kappa i,d)$ operaciones. La complejidad de un algoritmo crece monótonamente con respecto de sus argumentos (a menos que tenga complejidad constante, que no es el caso), de ahí podemos deducir que $O(K_p,d) > O(K_c,d') + O(\kappa i,d)$ cuando $K_p > K_c$ y $d' < d$.

Para estimar el ahorro que se tendría en un caso general, consideremos que el número de prototipos es el mismo para todas las clases y que el número de clases agrupadas en las ω_i es también igual y que la complejidad es lineal, Tendremos entonces $O(K,d) = Kd$, $K_p = pk$, $K_c = c\kappa$ y la complejidad en (5-7) es $O(pk,d) = pkd$, en (5-5), (5-6) $O(K_c,d')+O(\kappa,d)=\kappa cd'+\kappa d$.

Si fijamos p,d y k para un problema dado, $\kappa cd'+\kappa d$ es un conjunto de rectas en las variables d' y c. Para cada selección de d', tendremos c clases (c depende de la proyección que se tome y del conjunto de entrenamiento) y κ prototipos por cada clase.

5.2 Extensiones

El procedimiento descrito es válido si realizamos una nueva aplicación de una proyección sobre el espacio resultante. Nuevamente podemos aplicar lo discutido con un mapeo

$$L'(x):\mathbf{R}^{d'}\rightarrow\mathbf{R}^{d''} \qquad (5\text{-}8)$$

Y al espacio resultante podemos aplicarle un tercer mapeo, etc. Hasta que tengamos el caso en el que la proyección aplicada hace indistinguibles todas las clases. En este caso, tendríamos que realizar una cadena de aplicaciones del clasificador desde última proyección a la primera, y tendríamos un ahorro neto de operaciones en cada caso.

El álgebra de la justificación es la misma en cada aplicación del clasificador. Esto queda sumarizado en la figura 5.2 para el caso de estudio que nos ocupa. Observamos que este procedimiento define un grafo orientado, con una raiz (casi un árbol, execpto que puede tener ciclos). En la raiz

realizamos la primera clasificación que nos lleva a una de las ramas del árbol y ahí aplicamos otra clasificación hasta llegar a las hojas, en donde obtenemos la clasificación última que nos da la clase a la que pertenece el objeto desconocido (fig. 5.3).

5.3 Clasificación multiescala con k-vecinos

El clasificador de k-vecinos merece una discusión aparte, en este caso, el número de prototipos es el número de elementos del conjunto de entrenamiento. Si podemos descartar un conjunto de clases, no es necesario comparar a todos los elementos del conjunto de entrenamiento; sino solo a los que sabemos que pueden estar cercanos al objeto dado. Podemos mantener al mismo conjunto de entrenamiento en todas las proyecciones, el ahorro estará dado

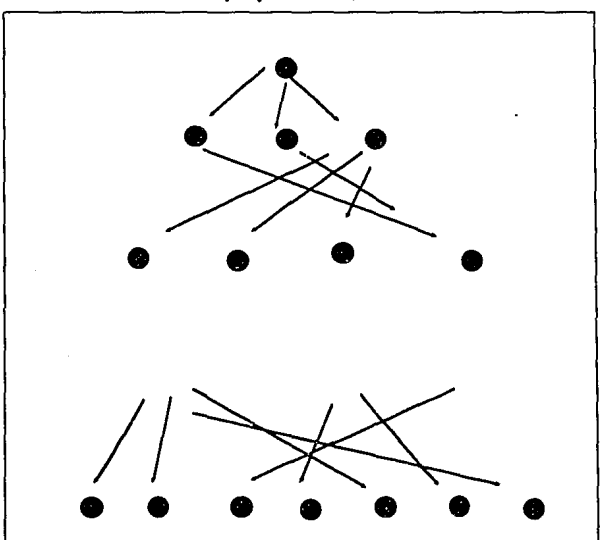


figura 5.2
Un árbol de decisión
Aplicaciones sucesivas del clasificador en distintos niveles

únicamente en función de la disminución de la dimensión de los vectores. Aún cuando habría una sobrecarga al calcular las clases indistinguibles, esto se hace fuera de línea y al tiempo de clasificación el número de cuentas se mantiene.

Adicionalmente, podemos hacer una depuración del número de elementos en el conjunto de entrenamiento para eliminar aquellos pares de vectores que estén mas cercanos que una cierta medida de tolerancia.

5.4 Clasificación multiescala para OCR

Describimos un experimento en la clasificación multiescala que ilustra el método de trabajo.

Es claro que para que funcione el algoritmo descrito en este capítulo, debemos tener un conjunto de entrenamien-

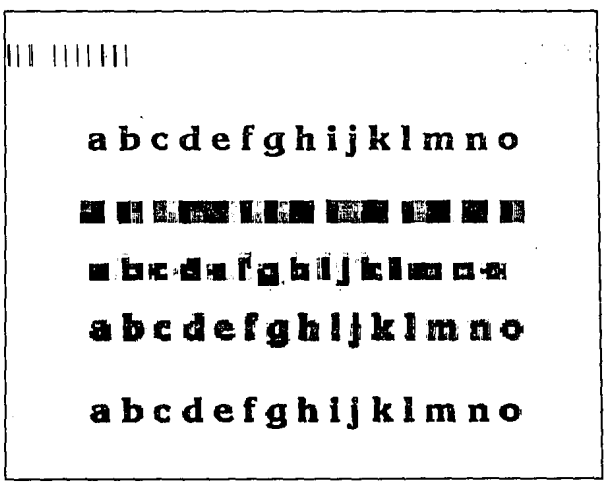


figura 5.3

La proyección empleada

Tomamos vistas del caracter, desde 1x1 (que equivale al promedio de gris en todo el caracter) hasta la resolución adecuada (8x8 en este caso). Se clasifica en etapas sucesivas hasta dejar fija la clase del caracter.

to y un problema que cumplan con las hipótesis del clasificador que se utilizará (i.e. si el problema es linealmente separable el clasificador será lineal, etc.). Como se discutió en el capítulo de clasificación, el problema de OCR no es linealmente separable cuando tomamos el caso de tipos de letra múltiples (multifont); por lo que se tomaron muestras de texto de un solo tipo de letra, de una hoja de texto y normalizados (obteniendo su promedio de gris) a matrices de 8x8 elementos. Los caracteres fueron tomados de distintos tamaños, de 8 a 18 puntos y digitalizados a 200 pixels por pulgada. Sabemos de antemano que el problema monofont es linealmente separable; por lo que tomamos un clasificador lineal para el caso de estudio.

descripción del método utilizado

Uno de los problemas que plantea el procedimiento multiescala, es encontrar el conjunto $W_c=\{\omega_1,...,\omega_c\}$, o sea las clases que se colapsan o que son indistinguibles en el espacio reducido. Discutiremos un procedimiento genérico que realiza esta tarea.

La proyección que se utiliza para pasar del espacio original al espacio reducido no preserva la geometría original. Para descubrir cuales clases se colapsaron podemos utilizar el mismo clasificador:

> Tomamos el conjunto de entrenamiento y obtenemos los parámetros de clasificación. Construimos la matriz A, con a_{ij}={1 si algún elemento de la clase i fué clasificado como perteneciente a la clase j; 0 de otra manera }.

Si A es la identidad significa que todos los elementos del conjunto de entrenamiento fueron clasificados correctamente; cada elemento distinto de cero fuera de la diagonal indica una clasificación cruzada.

La matriz A obtenida de esta manera describe claramente la geometría del problema siempre que el conjunto de entrenamiento sea representativo de los patrones que vamos a encontrar en la aplicación. Las clases colapsadas o clases de confusión serán los conjuntos obtenidos al colectar los unos de las columnas de la matriz. Por ejemplo, para la matriz

(5-9)

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Tendremos las clases de confusión a={ac}, b={be}, c={cd}, d={cd}, e={e}. Esto significa que alguna c fué clasificada como a; alguna e fué clasificada como b, alguna d fué clasificada como c, alguna c fué clasificada como d y ninguna letra fué confundida con la e.

Nótese como no es adecuado hacer simétrico este proceso;

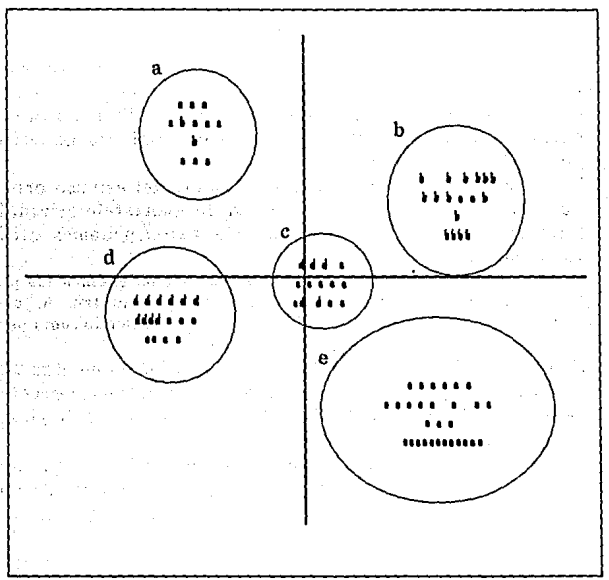


Figura 5.4

La matriz A y el conjunto de entrenamiento

Se puede establecer una correspondencia entre el conjunto de entrenamiento y la matriz de confusión.

si ninguna 'b' fué clasificada como 'e' , no es conveniente agrupar la clase de la 'e' con la de 'b'.

Con esta misma idea, se construyeron las matrices de confusión para el experimento realizado. Primero obtuvimos el conjunto de entrenamiento en las resoluciones 16x16, 8x8, 4x4, 2x2 y 1x1. Se aplicó el clasificador lineal a cada conjunto de entrenamiento obteniéndose las matrices A_{16},A_8,A_4,A_2 *y* A_1 que dan las siguientes clases de confusión:

	1	2	4	8	16
a	ABCDEFGHIJKLMNOPQRSTUVWXYZ	ACEFOQRSTVXYZ	A	A	A
b	Φ	BCEGHKLMQRUWZ	B	B	B
c	Φ	ACEGIMOQRSTVWXY	C	C	C
d	Φ	DIJ	D	D	D
e	Φ	Φ	CEO	E	E
f	Φ	AFIJLRSTX	FLQT	FT	F
g	Φ	Φ	GQ	G	G
h	Φ	Φ	BH	H	H
i	Φ	Φ	I	I	I
j	Φ	ADFJT	IJ	J	J
k	Φ	Φ	K	K	K
l	Φ	Φ	IL	IL	L
m	Φ	INR	M	M	M
n	Φ	Φ	N	N	N
o	Φ	Φ	CNO	O	O
p	Φ	ILMNPR	P	P	P
q	Φ	BCEGIKLMNOQSUVWZ	GQ	Q	Q
r	Φ	Φ	LR	R	R
s	Φ	Φ	ES	S	S
t	Φ	Φ	AT	T	T
u	Φ	Φ	U	U	U
v	Φ	Φ	VY	V	V
w	Φ	Φ	W	W	W
x	Φ	Φ	ILXZ	X	X
y	Φ	Φ	VY	Y	Y
z	Φ	Φ	IZ	Z	X

El símbolo Φ significa que ninguna letra fué clasificada

como perteneciente a la clase dada. Nótese que clasificar en A_1 no aporta ninguna información; no nos permite discriminar clases. Nótese también que no necesitamos poner representante en aquellas clases que no capturaron ninguna letra; por lo que en A_1 tenemos 1 clase, en A_2 tenemos 10 clases; y en A_4, A_8 y A_{16} tenemos 26 clases. En la tabla observamos el peor caso (en negritas) para la clasificación multiescala y en ese supuesto, tendríamos un 75% de ahorro con respecto a la clasificación tradicional. En el mejor caso (en itálicas) tendríamos un 90% de ahorro en cálculos. Esto lo obtenemos multiplicando el número de operaciones para una comparación (OP) por el numero de clases en cada caso.

a	ABCDEFGHIJKLMNOPQRSTUVWXYZ	ACEFOQRSTVXYZ	A	A	A
b	Φ	BCEGHKLMQRUWZ	B	B	B
c	Φ	IMOQRSTVWXY	C	C	C
d	Φ	*DIJ*	*D*	D	D
e	Φ	Φ	CEO	E	E
f	Φ	IJLRSTX	FLQT	FT	F
g	Φ	Φ	GQ	G	G
h	Φ	Φ	BH	H	H
i	Φ	Φ	*I*	*I*	I
j	Φ	DFJT	*IJ*	*J*	J
k	Φ	Φ	K	K	K
l	Φ	Φ	IL	IL	L
m	Φ	INR	M	M	M
n	Φ	Φ	N	N	N
o	Φ	Φ	CNO	O	O
p	Φ	ILHNPR	P	P	P
q	Φ	BCEGIKLMOQSUVWZ	GQ	Q	Q
r	Φ	Φ	LR	R	R
s	Φ	Φ	ES	S	S
t	Φ	Φ	AT	T	T
u	Φ	Φ	U	U	U
v	Φ	Φ	VY	V	V
w	Φ	Φ	W	W	W
x	Φ	Φ	ILXZ	X	X
y	Φ	Φ	VY	Y	Y
z	Φ	Φ	IZ	Z	X
OP	2	4	8	16	32
216	0	40	128	48	0
94	0	40	24	32	0

En la clasificación de un conjunto de 1000 vectores de prueba, obtuvimos un ahorro en tiempo efectivo de cómputo del orden de 35%; esto debido a que hay gastos adicionales (en segmentación, normalización, etc.) que representan la mayor parte de los cálculos.

Un punto importante es hacer notar que el ahorro es función de la partición que se haga, la cual de manera intuti-

tiva debe reflejar la estructura del patrón original.

A manera de conclusión, recomendaríamos un par de lineas de trabajo que amplirían el espectro de aplicación de este método de clasificación:

a)- Trabajar en la construcción de un clasificador multilineal (del estilo de k-medias), basado en la idea de clustering iterativo mejorando en cada paso la matriz de confusión (buscando acercarla a la identidad).

b)- Trabajar con clasificadores mas sofisticados como redes neuronales artificiales, (no queda claro como construir un clasificador que es adaptativo a un problema del que desconocemos la geometría; i.e. no podemos proponer una conectividad de la red a priori si desconocemos la geometría del conjunto de entrenamiento)

6

POSTPROCESAMIENTO

6.1 El problema

Ningún clasificador que se utilice nos dará certeza absoluta acerca de la identidad de los caracteres, siempre tendremos un margen de error. El origen de los errores puede localizarse en la etapa de preprocesamiento, cuando se segmentan los caracteres (el mas frecuente), o al clasificar. Es difícil mejorar las espectativas de buena clasificación solo en base a una buena función de discriminación y a un algoritmo de segmentación independientes, deben tomarse en cuenta las interrelaciones que existen en ambas etapas para llegar a un resultado satisfactorio.

En vista de lo anterior, es necesario tomar un remedial para mejorar las espectativas de certeza en el reconocimiento de una escena. Esta última etapa es llamada postprocesamiento.

El postprocesamiento consiste en un análisis de alto nivel que involucra interrelaciones entre *conjuntos de patrones identificados por el clasificador* y aceptados como probables; esto es, se analiza la escena completa y se verifica que ***haga sentido*** como un todo. Los errores cometidos al preprocesar, segmentar y clasificar los patrones de entrada son examinados en esta etapa de alto nivel.

Visto con esta óptica, existen dos tipos de problemas de reconocimiento de patrones; en el primero localizamos los problemas de jerarquía múltiple i.e. aquellos en donde el objetivo final corresponde a localizar patrones a diferentes niveles de granularidad; el segundo tipo son los problemas en donde los patrones identificados tienen sentido por sí

solos. Ejemplos para el primero son: reconocimiento de texto, voz etc.; para el segundo, el análisis cualitativo de muestras de materiales mediante métodos espectroscópicos, prospección geológica, percepción remota etc.

En problemas de jerarquía múltiple es posible hacer una validación por niveles de los resultados que entrega el clasificador; en el segundo caso, la clase que regresa el clasificador será el único parámetro que tendremos en la validación.

Como es evidente de esta discusión somera, el postprocesamiento tiene sentido cuando se aplica a problemas de jerarquía múltiple. En estos problemas es posible aplicar una validación contextual que rectifique (en el mejor de los casos) predicciones hechas por las etapas anteriores del problema. Por otro lado, cuando es el caso de un problema sin niveles jerárquicos la validación requiere de la ocurrencia de factores exógenos, los cuales pueden localizarse fuera del ámbito del reconocimiento de patrones. Estos factores pueden ser: el conocimiento que del problema que tenga un experto, pruebas adicionales que se hacen iterativamente al patrón dado, etc.

Cuando es el caso de problemas en los que el número de patrones contenidos en una escena es muy alto y el tiempo prfijado para la descripción completa de la escena es pequeño, el sentido común indica gastar poco esfuerzo (el mínimo posible) al clasificar los patrones de la escena y hacer una validación de alto nivel para corregir posibles errores (en el supuesto de que es mas barato hacer corrección contextual que hacer una clasificación sofisticada). Para el otro caso, vale la pena utilizar algoritmos de clasificación mas potentes aún cuando el número de operaciones sea costoso.

Para el caso particular que nos ocupa la velocidad de procesamiento es crucial, de ahí que sea importante dotar al sistema de un algoritmo barato para la clasificación. Esta velocidad de clasificación debe además estar parametrizada por el tipo de errores que se pueden cometer al clasificar, a saber: rechazos y clasificaciones cruzadas.

Como se discutió en el capítulo de clasificación, los rechazos son mas deseables que las clasificaciones cruzadas ya

que son detectables por un sistema manual o automático. Discutiremos en este capítulo técnicas para detectar algunas clasificaciones cruzadas en un sistema de OCR.

detección de errores

Supongamos que tenemos una página de texto entregada por el clasificador; buscamos detectar los errores cometidos y marcar todas la palabras que tengan errores. Para esto, el caso interesante es cuando el error que se comete es de clasificación cruzada (de otra manera el error ya está localizado).

Caso 1: La palabra no existe en el español

Para determinar si en una palabra hay una clasificación cruzada en este caso, basta con consultar en un diccionario; si no encontramos la palabra la marcamos.

Caso 2: La palabra existe en el español

Este es un punto delicado; dado que la palabra existe, una consulta al diccionario no es útil. Para detectar este error, es necesario hacer un análisis sintáctico de una frase completa para decidir que esa palabra está equivocada. Este análisis es caro en tiempo de cómputo y a la fecha, no hay ningún sistema que lo lleve a cabo con resultados satisfactorios.

Hay un tercer caso, cuando los símbolos de que consta el sistema abarca mas que alfanuméricos, la detección de clasificaciones cruzadas simbolo-alfanumérico, alfanumérico-símbolo es casi trivial y no la discutiremos.

Discutiremos como construir un método barato de consulta en un diccionario para detectar errores de clasificación cruzada; así como una propuesta para localizar errores sintácticos.

corrección de errores

Dada la naturaleza del lenguaje, la corrección automática de errores es solo posible en un número limitado de casos. Si el error es de tipo sintáctico, es casi imposible (con la herramienta teórica actual) encontrar cual es la frase correcta; para ello sería necesario un análisis semántico de la frase para encontrar la palabra que hace sentido, lo cual es un problema abierto en procesamiento del lenguaje natural. Si, por otro lado, una palabra específica no se en-

cuentra en el diccionario, existen generalmente muchas palabras del español *cercanas* a la palabra equivocada, de donde plantear un método automático de corrección, dista mucho de ser realista. La alternativa es construir un sistema interactivo que permita desplegar una serie de alternativas en una ventana, de donde se pueda seleccionar con facilidad la palabra adecuada.

5.1 Detección de errores en palabras

Los algoritmos tradicionales de búsqueda en árboles binarios o árboles-b requieren de la construcción de un árbol de búsqueda en el que hay que emplear memoria adicional a la tabla de datos misma. Esto representa un costo muy alto si se quiere mantener el diccionario en memoria. El tiempo de búsqueda en árboles es logarítmico con respecto al número de entradas que tenga la tabla. Propondremos un algoritmo de hash externo que balancea adecuadamente el tiempo empleado en la consulta y la cantidad de memoria necesaria para la búsqueda. La suposición principal es que la mayoría de las veces la palabra *sí* se va a encontrar en el diccionario.

funciones de hash

Una búsqueda secuencial en una tabla tiene un tiempo lineal de ejecución, si el tamaño de la tabla es N, debemos hacer cuando mas N comparaciones para decidir si la palabra se encuentra. Esta es una cota muy alta considerando sobre todo que el tamaño de un diccionario de español es muy grande.

La idea de una función de hash consiste en tener una idea aproximada de donde se localiza la palabra y una vez localizada la región hacer una búsqueda de secuencial corta (del orden de 2 a 4 elementos). Para lograr esto construimos una tabla **h** de tamaño fijo **p** un número primo (por razones que veremos mas adelante) y distribuir adecuadamente todos los elementos de la tabla original en las entradas de la nueva tabla **h**. Las entradas de **h** las llamaremos buckets. La función de hash, entonces, localiza el bucket en el que se almacenó la palabra buscada y emplea una búsqueda secuencial dentro de él para decidir si la

palabra se encuentra ahí.

1	uno
2	dos
3	tres
4	cuatro
5	cinco
6	seis
7	siete
8	ocho
9	nueve
10	diez
11	once
12	doce
13	trece
14	catorce
15	quince
16	dieciseis
17	dieisiete
18	dieciocho
19	diecinueve
20	veinte

Figura 6.1a
Una tabla de tamaño veinte, con entradas alfanuméricas.

1	uno	trece	diecinueve		
2	siete	nueve	dieciseis	ocho	once
3	dos	diecisiete	cinco		
4	doce	catorce	tres	dieciocho	
5	diez	cuatro	seis	quince	veinte

Figura 6.1b
La tabla anterior redistribuida, de tal modo que primero localizamos un bucket y después buscamos dentro de él la palabra adecuada

Las condiciones que debe cumplir la función de hash son 1) Distribuir las entradas de la tabla de tal modo que cada bucket contenga aproximadamente la misma cantidad de elementos 2) Localizar sin error el bucket en donde se puede encontrar la palabra.

Para cumplir con la segunda condición hacemos una aplicación de la función de hash a cada uno de los elementos de la tabla original y los colocamos en el bucket que indica es decir:

h[hash(t[i])] = t[i] i=1,..,N

de este modo, todos los elementos de la tabla t tendrán una entrada en la tabla h. Si x es una palabra que buscamos, basta aplicar la función hash(x) y buscar en el bucket que se indica. Si la palabra no se encuentra en el bucket, entonces no se encuentra en la tabla.

Para que se distribuyan uniformemente en todos los buckets los elementos de la tabla debemos tener una tabla de tamaño p un primo. Esto en razón de que la función de hash es en cierta medida una operación módulo sobre el tamaño de la tabla original. Si el tamaño de la tabla es un producto de primos, entonces habrá buckets vacíos y otros que se llenarán rápidamente. El tiempo promedio de búsqueda con funciones de hash es constante (no depende del número de elementos de la tabla original) y proporcional al tamaño de cada bucket.

El problema de aplicar directamente esta técnica para buscar en diccionario es que la cantidad de memoria requerida para mantener la tabla en RAM es alta, porque además debemos mantener una lista ligada dentro de cada bucket. Es por ello que formulamos una implementación alternativa conocida como hash externo.

hash externo

La idea esencial de esta técnica es la de construir una tabla que contenga las direcciones en disco (el offset a partir de la dirección inicial de un archivo) de cada una de las palabras. De esta manera, tenemos un balance adecuado entre la cantidad de memoria RAM utilizada y la velocidad de consulta. Una consulta al diccionario consiste en calcular una función de hash, que nos regresa un índice,

que es la entrada de una tabla que contiene la dirección física de una palabra en un archivo de disco (fig. 2) . La rutina de búsqueda sería

```
BOOLEAN busca( char * word, FILE *file, t_index *t)
{
  t_index index;
  char comp[MAX_WORD];

    index = t[hash(word)];
    fseek(file, index, START);
    fscanf(file,"%s",comp);
    return strcmp(comp,word);
}
```

Para construir la tabla de índices, calculamos la función

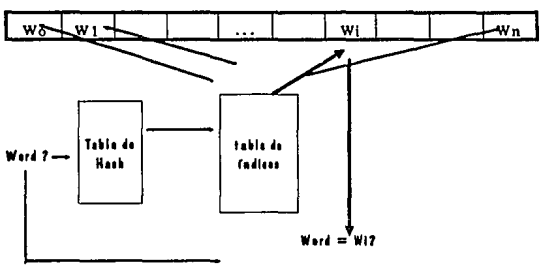


Figura 6.2
Búsqueda con hash externo

de hash de las palabras en el archivo y en el bucket correspondiente colocamos su posición. Nótese que este esquema tiene la ventaja de no requerir ningún ordenamiento de las palabras; se pueden agregar nuevas palabras al diccionario pegándolas al final del archivo y agregando su

índice en la tabla, vía la función de hash.
La instrumentación que se hizo facilita aún mas la búsqueda almacenando de manera separada las palabras por la letra con la que comienzan. De esta manera tendremos archivos a.dic, b.dic, etc. y relacionados con ellos los archivos a.hsh, b.hsh, etc. en donde guardamos las tablas precalculadas bajo la función de hash dada. Para que el programa comience a funcionar cargamos cada una de las tablas *.hsh en memoria y aplicamos la función de búsqueda dependiendo de la primera letra de la palabra de entrada.
Los experimentos realizados para las búsquedas exitosas y no exitosas revelan que el tiempo requerido para una consulta promedio es bajo. Para primos del orden de 50,000, no tendremos (casi) colisiones en la tabla de hash, por lo que en promedio estaremos haciendo un acceso a disco para localizar una palabra.

automata finito de secuencias

Este algoritmo es razonablemente rápido cuando tenemos un disco duro de tamaño adecuado y rapidez para acceder a la información; pero pierde eficiencia a medida que los accesos a disco son costosos. Para mejorar el algoritmo, discutiremos la construcción de un autómata finito que hace pre-fetch de las palabras a buscar.
Buscaremos implementar un algoritmo que nos permita localizar combinaciones razonables de vocal-consonante, consonante-vocal, vocal-vocal y consonante-consonante.
Definiremos para ello un autómata con un número fijo de estados. Este nos dará un mapeo inyectivo, de tal forma que si el autómata rechaza la palabra ésta no se encontrará en el diccionario; pero si la acepta hay que hacer una búsqueda todavía.
Sea $M=(K,\Sigma,q_0,\delta,K)$ un autómata finito no determinístico, con K un conjunto de $k|\Sigma|$ estados
$\{[q_0a],[q_0b],\ldots[q_0\tilde{n}],[q_1a],\ldots,[q_k\tilde{n}]\}$
Σ el alfabeto de entrada que consta de las letras minúsculas, letras acentuadas y la ñ. $|\Sigma|$ el número de letras del alfabeto. q_0 el estado inicial y K el conjunto de estados finales. $\delta: K\times\Sigma\rightarrow K$ la función de trancisión definida iterativa-

BIBLIOGRAFIA

Libros:

Pattern Recognition By Self-organizing Neural Networks *Carpenter & Grossberg*, editors. The MIT Press, 1991.

Computer Image Processing and Recognition. *Ernest L. Hall.* Academic Press, 1979.

Heuristics, *Judea Pearl.* Addison-Wesley. 1984

Dynamic, Genetic, and Chaotic Programming. *Branco Soucek and The Iris Group.* Wiley & Sons. 1992.

Pattern Recognition. *Robert Schalkoff.* Wiley & Sons. 1992.

Digital Filters. *R.W. Hamming.* Prentice Hall. 1989.

Proceedings of the IFAC Symposium. Pergamon Press, Elmsford NY 1984.

Algoritms in C. *Robert Sedgewick.* Addison Wesley. 1990.

Data Structures and Algorithm Analisis in C. *Mark Allen Weiss.* Benjamin Cummings. 1992.

Fundamental Algorithms, *Donald E. Knuth.* Addison Wesley. 1981

Sorting and Searching. *D. E. Knuth. Addison* Wesley. 1981

Seminumerical Algorithms. *D. E. Knuth.* Addison Wesley. 1981

New C Primer Plus. *Waite & Pratta.* SAMS. 1990

Articulos

High Accuracy character recognition using Fourier and Topological descriptors. *Shriphar, M; Badreldin A.* Pattern Recognition 17, 5 (1984)

Image Thresholding for optical character recogni-

mente:
Inicialmente δ no está definida para ninguna palabra, por cada $w=a_0a_1...a_l$ que se encuentra en el diccionario hacemos $\delta([q_ia_i],a_i)=[q_{i+1}a_{i+1}]$ y $\delta(q_0,a_0)=[q_0a_0]$
Para ilustrar que se trata de un mapeo inyectivo y no biyectivo (lo que sería ideal), la función de transición en la figura 6.3

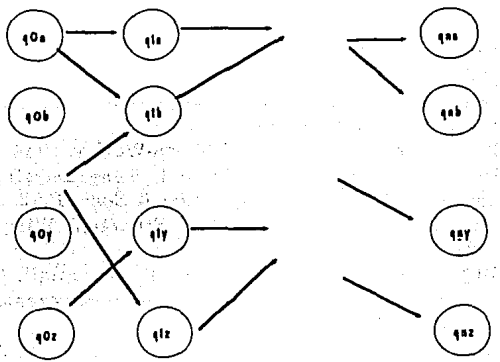


figura 6.3
Construcción del autómata finito de secuencias

Otros algoritmos

La potencia de un algoritmo de búsqueda en diccionario consiste en localizar palabras incompletamente especificadas, que no es el caso que describimos en las secciones anteriores. Una implantación de árboles PATRICIA sería de mas utilidad en aplicaciones reales de OCR, dado que en el se pueden hacer búsquedas de palabras incompletamente especificadas.
Un algoritmo de búsqueda de palabras incompletamente especificadas tiene la ventaja de corregir inclusive pala-

bras que fueron mal escritas en el original. Como consideración de diseño, es adecuado hacer énfasis en una etapa robusta de postprocesamiento i.e. que nos permita hacer correcciones de letras mal clasificadas a partir de la información contextual dentro de cada palabra.

Sería interesante construir un algoritmo que realizara búsqueda en diccionario para una palabra especificada como un lista de prioridades (ver capítulo de segmentación). A manera de ejemplo, consideremos la palabra {(ce)(li)(ad)(vu)(ce)}, donde las letras entre paréntesis representan las alternativas que entregaron las etapas de segmentación y clasificación, en orden de calificación; de esta forma en el primer par c es la letra mas probable, en el segundo la l, etc. Con esta información, la palabra "clavc" es la que se obtiene considerando la letra mas probable en cada sublista; sin embargo no existe. El algoritmo haría una búsqueda de la palabras mas probables, seleccionando aquellas que se encuentren en el diccionario, es decir, tendríamos las palabras "clave", "clauc", "claue", "cldue", etc. como alternativas y "clave" como la mas probable, dado que se encuentra en el diccionario.

6.3 Detección de errores en frases

El procesamiento del lenguaje natural es un tópico complicado que merece la atención de un gran número de investigadores en todo el mundo; sin embargo, ha resultado ser todo un reto y a la fecha no podemos hablar de que el problema se encuentre resuelto completamente. La detección de posibles errores sintácticos tiene una connotación restringida, es decir, es un problema menor dentro del problema general. Inspirado en el procedimiento descrito en el apartado anterior, discutiremos una técnica para detectar errores sintácticos en frases.

Es posible agrupar a las palabras de un lenguaje natural en clases. Cada clase corresponde a un tipo de palabra; así tendremos los sustantivos, adjetivos, pronombres, verbos, preposiciones, adverbios (de tiempo, de modo, etc), tropos, cuantificadores, conectivos etc.

La categorización del diccionario puede darse en una cierta granularidad, o sea, podemos descomponer a los sustan-

tivos, verbos, etc. en subclases de un tamaño tan fino como sea necesario.

Supongamos que tenemos un conjunto de entrenamiento que consiste de un libro, y cada frase en el libro es una cadena ejemplo. Con base en este conjunto, debemos ser capaces de calcular para una secuencia que no se encuentre en el diccionario su plausibilidad; es decir, la probabilidad de que la cadena está bien formada.

Estos requerimientos definen de manera natural una seria de tiempo; es decir, para una secuencia $w=a_0a_1...a_s$, debemos calcular las probabilidades de que a_i corresponda a la secuencia dado que ocurrieron $a_0a_1...a_{i-1}$ para cada i (para validar las primeras podemos pensar por simetría que las dadas son las últimas).

Este procedimiento permitiría encontrar, con cierta probabilidad, palabras mal colocadas en frases.

tion and other applications requiring character image extraction. *Whited, J.M. Rohdere G. D.*; IBM Journal of Research and development 27, 4 (1983)

A simple learning decision Algorithm for character recognition and pattern clasification. Pattern Recognition 10, 2. (1978) pp. 99-104.

OCR: rediscovering and old technology. *Stibben S.* Infosystems 27, 6 (jun 1980)

A matrix approach to character recognition. *Fritszch K.* Pattern Recognition 11, 3 (1979)

Optical character recognition based on analog preprocesing and automatic feature extraction. *Lashas A. et all.* Coputer Vision, Graphics and Image Procesing 32,2 (nov 85)

Character recognition and imaging come to desktop. *Williams T.* Comput. Des. 24,9 (1985)

A new character recognition scheme with lower ambiguity and higer recognizability. *Wang P. S.* Pattern Recognition Letters 3, 6 (1985)

Psychophysical techniques for investigating the disctinctive feature of letters. *R. J. Shillman et all.* Int J. Man-Machine Studies (1976).