Universidad Politécnica de Valencia
Facultad de Informática

Aprendizaje de modelos discretos semiparamétricos y su aplicación a la clasificación de texto

Proyecto Final de Carrera

Realizado por
David Vilar Torres

Dirigido por los doctores
D. Alfons Juan Ciscar y D. Enrique Vidal Ruiz

Valencia, junio de 2003

A mi padre, a quien me quedó mucho por decir y, sobre todo, por agradecer.

Agradecimientos

Me gustaría mostrar aquí mi agradecimiento a diversas personas que de una forma u otra me han ayudado a lo largo de toda la carrera.

En primer lugar me gustaría agradecer a Alfons Juan y Enrique Vidal que se ofrecieran a dirigir este proyecto, su apoyo a lo largo del curso y las puertas que me han abierto para el futuro.

A Mª José Castro y Emilio Sanchis les quiero agradecer así mismo el apoyo que me han dado a lo largo de este año, las oportunidades que me han ofrecido y la paciencia que han tenido conmigo.

A mis hermanos Juan Miguel y Cristina, que al haber elegido yo su misma carrera, me han dado siempre buenos consejos y me han prestado su ayuda cuando la necesitaba.

Y en general a mi familia, a mi hermano Carlos y a mi madre, y a mis compañeros de clase y amigos, que sé que siempre podré contar con ellos.

D.V.T.
Valencia, junio de 2003

II

Índice general

Índice de figuras

Índice de tablas

Índice de tablas

En cualquier teoría particular sólo hay de ciencia real lo que haya de matemáticas.

IMMANUEL KANT

En la medida en que las proposiciones de las matemáticas dan cuenta de la realidad, no son ciertas; y en la medida en que son ciertas, no describen la realidad.

ALBERT EINSTEIN

2

CAPÍTULO 1

Preliminares

1.1. Introducción

El *reconocimiento de formas* es una disciplina fuertemente vinculada a la informática, que estudia cómo construir sistemas automáticos capaces de emular aspectos "perceptivos" propios del comportamiento humano, por lo general relacionados con el habla o la visión.

El paradigma clásico en el que se sustenta esta disciplina es el *paradigma de la clasificación*. De acuerdo con el mismo, un sistema de reconocimiento de formas puede verse como una "caja negra", tal y como se ilustra en la figura 1.1. La entrada del sistema es una señal obtenida mediante transductores que miden el objeto a reconocer. La salida no es más que una etiqueta de clase perteneciente a un conjunto finito de etiquetas previamente fijado. El objetivo típico es minimizar la *probabilidad de error* en la clasificación o, más generalmente, el *riesgo total*.

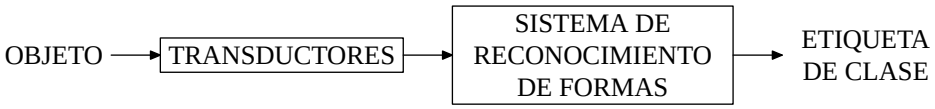

FIGURA 1.1: Modelo "caja negra" de un sistema de reconocimiento de formas.

El modelo "caja negra" anterior puede refinarse como se ilustra en el diagrama de bloques de la figura 1.2. El primer bloque del sistema se denomina *preproceso*. Este bloque se encarga de la *adquisición y mejora* (filtrado) de la señal. A continuación la señal mejorada atraviesa un segundo bloque que llamamos *extracción de características*. Como su nombre sugiere en este bloque se extrae la información relevante para la clasificación del objeto, la cual se expresa mediante un conjunto de descriptores en un formato prefijado, dependiente del sistema. Dos

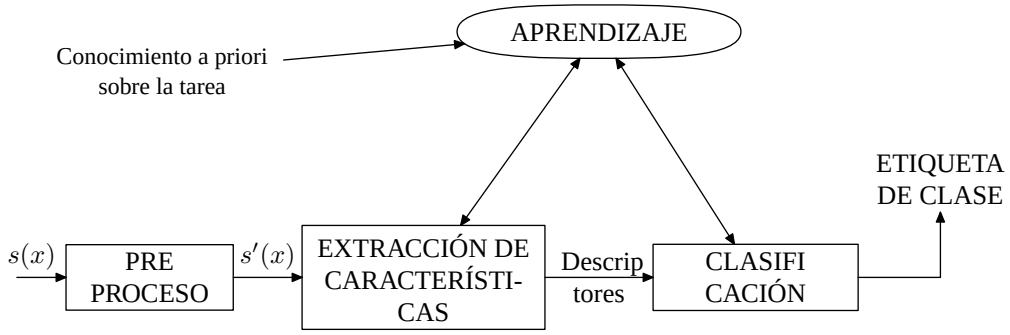

FIGURA 1.2: Diagrama de bloques de un sistema de reconocimiento de formas.

formatos habituales para la representación de objetos son el *vector de caracteríticas* (numéricas) y la *cadena de símbolos* (caracteres). Por último, los objetos representados en el formato normal del sistema son etiquetados en un tercer bloque que llamamos *clasificación*. El diseño de este bloque, al igual que el de extracción de características, se llevará a cabo a partir de nuestro conocimiento a priori sobre la tarea, en una etapa de *aprendizaje*.

Es importante distinguir entre la etapa de aprendizaje y la etapa operativa del sistema, a la cual nos referimos como *reconocimiento* (figura 1.3). En reconocimiento ya hemos diseñado e implementado el sistema, por lo que está disponible para su utilización por parte del usuario. Éste quiere un sistema eficaz además de eficiente, capaz de poder adaptarse a un entorno concreto, tal vez cambiante. En aprendizaje, sin embargo, nos hallamos en el laboratorio, con recursos computacionales altos que empleremos para escoger y ajustar un método de extracción de características y otro de clasificación. Al aprendizaje del primer método le llamamos *selección de características* mientras que al aprendizaje del segundo nos referimos como *diseño del clasificador*. Aunque ambos son importantes, el diseño del clasificador es, con diferencia, la parte del aprendizaje que recibe mayor atención.

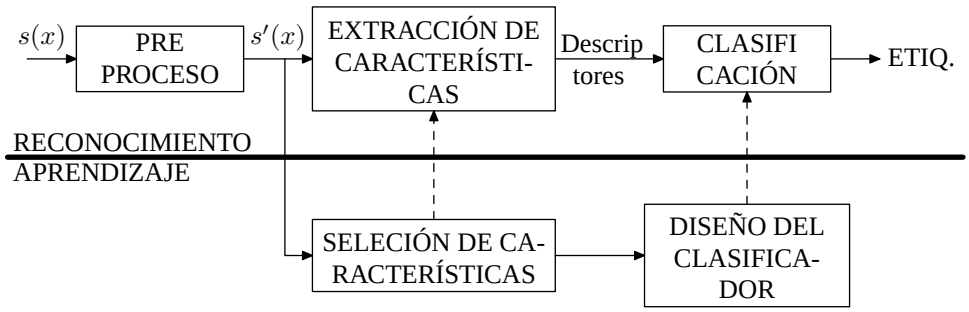

FIGURA 1.3: Etapas de un sistema de reconocimiento de formas.

El diseño del clasificador se realiza *inductivamente*, a partir de ejemplos o *muestras*. Depen-

diendo de estas muestras podemos distinguir dos tipos básicos de aprendizaje. Si las muestras están correctamente etiquetadas estamos hablando de aprendizaje *supervisado*, en contraposición al aprendizaje basado en muestras sin etiqueta de clase, al que nos referimos como *aprendizaje no supervisado* o *clustering*.

Se pueden distinguir dos aproximaciones principales al reconocimiento de formas: la *geométrica* (o *estadística*) y la *sintáctica* (o *estructural*). La primera, posiblemente la más extendida, se basa en la teoría estadística de la decisión mientras que la segunda se basa en la teoría de lenguajes formales.

En el marco de la aproximación estadística, el presente proyecto centrará su atención en el aprendizaje (supervisado y no supervisado) de modelos probabilísticos semiparamétricos y su aplicación a clasificación de textos. Se estudiará la eficacia del método aplicado a distintos corpus de datos, así como la influencia de los parámetros que gobiernan el comportamiento de los algoritmos y la importancia que el proceso de suavizado tiene sobre los resultados finales.

1.2. Teoría de la decisión de Bayes

En un sistema tradicional de reconocimiento de formas disponemos de un *conjunto de entrenamiento* $\{(\boldsymbol{x}_i, c_i)\}_{i=1}^{N}$, donde $\boldsymbol{x}_i \in E$ es la imagen de un objeto en un espacio de representación E adecuado, y $c_i \in \mathcal{C}$ es la *etiqueta de clase*, siendo el conjunto $\mathcal{C}$ finito y conocido a priori. A partir de este conjunto deseamos inferir un modelo con el que poder clasificar nuevas muestras, de las que no conocemos la etiqueta de clase asociada, en una de las clases del conjunto $\mathcal{C}$. Es decir, buscamos un función $G : E \longrightarrow \mathcal{C}$.

Denotaremos por $p(c)$ la (función de) probabilidad a priori de la clase c es decir, la proporción de muestras pertenecientes a la clase c respecto del total[1]. Mediante $p(\boldsymbol{x}|c)$ denotamos la distribución de probabilidad de las muestras pertenecientes a la clase c o probabilidad condicional y $p(c|\boldsymbol{x})$ representa la probabilidad condicional de la clase c dada la muestra $\boldsymbol{x}$, es decir, la probabilidad de que habiendo observado la muestra $\boldsymbol{x}$ ésta pertenezca a la clase c. Estas cantidades están relacionadas mediante la *regla de Bayes*

$$p(c|\boldsymbol{x}) = \frac{p(c)p(\boldsymbol{x}|c)}{p(\boldsymbol{x})} \tag{1.1}$$

donde

$$p(\boldsymbol{x}) = \sum_{c \in \mathcal{C}} p(c)p(\boldsymbol{x}|c) \tag{1.2}$$

es la probabilidad incondicional de observar la muestra $\boldsymbol{x}$. La ecuación 1.1 tiene una gran utilidad a la hora de llevar a cabo las tareas de clasificación, como veremos seguidamente.

[1]Sería más correcto definir una variable aleatoria C y expresar la probabilidad a priori de la clase c como $p(C = c)$. Sin embargo esta notación, aunque formalmente más correcta, habitualmente complica (visualmente) las ecuaciones de forma innecesaria, por lo que adoptamos la forma expuesta en el texto.

Consideremos ahora la probabilidad de error al clasificar una muestra $\boldsymbol{x}$ en la clase c. Dada la probabilidad real $p(c|\boldsymbol{x})$ de que la muestra $\boldsymbol{x}$ pertenezca a la clase c, la probabilidad de error es

$$p_c(\text{error}|\boldsymbol{x}) = 1 - p(c|\boldsymbol{x}) \tag{1.3}$$

Nuestro objetivo es minimizar la probabilidad de error al clasificar una muestra $\boldsymbol{x}$, de ahí

$$p(\text{error}|\boldsymbol{x}) = \min_{c\in\mathcal{C}} p_c(\text{error}|\boldsymbol{x}) = 1 - \max_{c\in\mathcal{C}} p(c|\boldsymbol{x}) \tag{1.4}$$

es decir, minizamos la probabilidad de error al clasificar en la clase c que tenga mayor probabilidad a posteriori. Podemos definir entonces nuestra función de clasificación como

$$G(\boldsymbol{x}) = \operatorname*{argmáx}_{c\in\mathcal{C}} p(c|\boldsymbol{x}) \tag{1.5}$$

A este clasificador se le denomina *clasificador de mínimo riesgo* o *clasificador de Bayes*.

Utilizando la regla de Bayes y propiedades básicas del operador $\operatorname{argmáx}$ se puede verificar que

$$\begin{aligned} G(\boldsymbol{x}) &= \operatorname*{argmáx}_{c\in\mathcal{C}} p(c|\boldsymbol{x}) = \operatorname*{argmáx}_{c\in\mathcal{C}} \frac{p(\boldsymbol{x}|c)p(c)}{p(\boldsymbol{x})} = \operatorname*{argmáx}_{c\in\mathcal{C}} p(\boldsymbol{x}|c)p(c) \\ &= \operatorname*{argmáx}_{c\in\mathcal{C}} \left(\log p(\boldsymbol{x}|c) + \log p(c)\right) \end{aligned} \tag{1.6}$$

Esta propiedad tiene una gran importancia práctica, por un lado porque habitualmente resulta más fácil estimar las probabilidades $p(c)$ y $p(\boldsymbol{x}|c)$ y por otro lado, ya que las probabilidades son de un orden de magnitud considerablemente pequeño, al trabajar con logaritmos podemos evitar problemas de *underflow*. Conviene así mismo considerar la función de clasificación como la maximización de una familia de *funciones discriminantes* $\{g_c(\boldsymbol{x})\}_{c\in\mathcal{C}}$ donde $g_c(\boldsymbol{x}) = p(c|\boldsymbol{x})$ o equivalentemente $g_c(\boldsymbol{x}) = \log p(c) + \log(p(c|\boldsymbol{x}))$.

1.3. Aprendizaje por máxima verosimilitud

Dada la regla de clasificación expuesta en la sección anterior, nuestra tarea consiste en estimar las probabilidades a priori $p(c)$ de cada una de las clases $c \in \mathcal{C}$, además de las distribuciones de probabilidad $p(\boldsymbol{x}|c)$. Buscamos, pues, un vector de parámetros $\boldsymbol{\theta} = (p(1), \ldots, p(C); \boldsymbol{\theta}')$, siendo $\boldsymbol{\theta}'$ el vector de parámetros de los que dependen las distribuciones de probabilidad $p(\boldsymbol{x}|c)$ y C el cardinal del conjunto $\mathcal{C}$. Dado un conjunto de entrenamiento $\{(\boldsymbol{x}_n, c_n)\}_{n=1}^N$ podemos entonces definir la función de *verosimilitud* como la probabilidad de haber observado las muestras de entrenamiento con los parámetros $\boldsymbol{\theta}$, es decir

$$L(\boldsymbol{\theta}) = p((\boldsymbol{x}_1, c_1|\boldsymbol{\theta}), \ldots, (\boldsymbol{x}_n, c_n|\boldsymbol{\theta})) \tag{1.7}$$

Asumiendo que las observaciones $\{(\boldsymbol{x}_n|c_n)\}_{n=1}^N$ son estadísticamente independientes obtenemos

$$L(\boldsymbol{\theta}) = \prod_n p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}) = \prod_n p(c_n|\boldsymbol{\theta})p(\boldsymbol{x}_n|c_n;\boldsymbol{\theta}) \tag{1.8}$$

Nuevamente trabajamos con logaritmos, obteniendo así la denominada función de *log-verosimilitud*

$$\log L(\boldsymbol{\theta}) = \sum_n \left(\log p(c_n|\boldsymbol{\theta}) + \log p(\boldsymbol{x}_n|c_n;\boldsymbol{\theta})\right) \tag{1.9}$$

El criterio que seguiremos para estimar el vector de parámetros $\boldsymbol{\theta}$ será la maximización de la función de verosimilitud o, equivalentemente, la log-verosimilitud. Nuestra estimación será por tanto

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmáx}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \operatorname*{argmáx}_{\boldsymbol{\theta}} \log L(\boldsymbol{\theta}) \tag{1.10}$$

Al realizar esta maximización, el vector de parámetros $\boldsymbol{\theta}$ no puede variar arbitrariamente dado que tiene que estar sujeto a una serie de restricciones, como por ejemplo que los $p(c)$ tienen que ser efectivamente una función de probabilidad y por tanto

$$\sum_{c\in\mathcal{C}} p(c) = 1 \tag{1.11}$$

y otras restricciones impuestas por las funciones de probabilidad condicionales $p(\boldsymbol{x}|c)$.

Dado que optimizamos la función de verosimilitud para el conjunto de entrenamiento podemos encontrarnos con el problema de que nuestro modelo aprende "demasiado bien", memorizando las características propias de este conjunto, sin llegar a generalizar, con lo que las nuevas muestras que llegan al sistema no son clasificadas correctamente. Este efecto se conoce como *sobreentrenamiento* y tenemos que aplicar métodos para perturbar los parámetros que, aunque actuando contra el criterio de máxima verosimilitud, aumentan la precisión del clasificador.

CAPÍTULO 2

La tarea

Nuestra tarea consistirá en utilizar modelos paramétricos y semiparamétricos a la clasificación de texto. Ésta es un área de investigación relativamente nueva y que en los últimos años está adquiriendo una atención cada vez mayor, debido en gran parte al auge de internet. Dada la gran cantidad de información disponible en la red interesa disponer de métodos eficaces para la clasificación de documentos, de tal forma que se pueda mejorar la efectividad de buscadores de páginas web o filtros de mensajes de correo electrónico.

Para poder comparar la efectividad de los distintos métodos en este proyecto utilizamos tres corpora distintos, de cada uno de los cuales se incluye una breve introducción en este capítulo.

2.1. EuTrans

El corpus EuTrans se compone de un subconjunto del corpus utilizado en el proyecto de traducción automática EuTrans ESPRIT. En este proyecto se desarrolló un traductor español-inglés de voz a voz en el dominio restringido de un viajante (turista) visitando un país extranjero. Para la tarea se definieron distintos escenarios de traducción y en este trabajo nos limitamos a situaciones de comunicación en la recepción de un hotel.

Para obtener un corpus de esta tarea, se reunieron distintas guías de viaje y se seleccionaron las frases que se acomodaban al escenario que acabamos de definir. Con el fin de limitar la complejidad de la tarea sólo se seleccionaron 16 subdominios, con lo que se consiguió un "corpus semilla" de tamaño reducido que se extendió a un conjunto mayor de frases utilizando un procedimiento semiautomático. Este proceso se llevó a cabo de forma independiente por cuatro personas, de las cuales cada una debía seleccionar aquellas pertenecientes a un subconjunto de los subdominios. Cabe destacar que estos subconjuntos asociados no eran disjuntos. Como resultado se obtuvieron cuatro clases de frases, etiquetados como A, F, J y P, en función de la persona

Clases				Subdominios	
A	F	J	P	Nº	Descripción
√	√			1	Notificación de una reserva previa
√				2	Preguntar sobre habitaciones
√				3	Mirar las habitaciones
√	√			4	Pedir una habitación
√				5	Firmar en el registro
√				6	Quejas acerca de las habitaciones
√				7	Cambio de habitaciones
	√			8	Pedir el servicio despertador
	√			9	Pedir las llaves
	√	√		10	Pedir que lleven el equipaje
		√		11	Informar de la partida
			√	12	Pedir la factura
		√	√	13	Preguntar acerca de la factura
			√	14	Quejarse sobre la factura
			√	15	Pedir un taxi
√	√	√	√	16	Frases generales

TABLA 2.1: Subdominios de la tarea EuTrans y su asignación a las clases.

encargada de reunir las frases. La tabla 2.1 muestra los 16 subdominios y la asignación de las clases. Un ejemplo de las frases del corpus es:

- Reservé una habitación individual y tranquila con televisión hasta pasado mañana.
- Por favor, prepárenos nuestra cuenta de la habitación dos veintidós.

Las clases definidas en este corpus resultan un tanto artificiales, pero debido a su manejable tamaño y a su composición claramente definida, este corpus es muy apropiado para realizar experimentos.

2.1.1. Preproceso y partición

El corpus utilizado en los experimentos se compone de 8000 frases de la tarea descrita, 2000 de cada una de las clases. La primera mitad se dedicó al conjunto de entrenamiento y la segunda al conjunto de test. Se eliminaron las palabras que aparecen una única vez, reduciéndose el vocabulario a 614 palabras. La longitud media de los documentos es de 7.9 palabras.

2.1.2. Resultados anteriores

Dado que este corpus es relativamente artificial existen pocos resultados que hagan referencia a estos datos. En [JV02] se utilizan mixturas de distribuciones de Bernoulli para esta tarea, consiguiendo una tasa de error mínima del $1.5\,\%$. Nosotros utilizaremos una aproximación similar utilizando mixturas de multinomiales en el capítulo 5.

2.2. WebKB

Este conjunto de datos contiene un total de $8\,282$ páginas html extraídas de distintos departamentos de informática en Enero de 1997 por el proyecto "World Wide Knowledge Base (Web→KB)" del grupo "CMU text learning group" ([Gro]). Las páginas se dividieron manualmente en las siguientes categorías (entre paréntesis se muestra el total de páginas de cada categoría y en cursiva la traducción del nombre de cada una de la clases):

- student ($1\,641$) [*estudiante*]
- faculty ($1\,124$) [*facultad*]
- staff (137) [*empleados*]
- department (182) [*departamento*]
- course (930) [*cursos*]
- project (504) [*proyectos*]
- other ($3\,764$) [*otros*]

La clase "otros" reúne todas aquellas páginas que no se pueden clasificar como la página principal de cada una de las otras categorías. Para cada clase se reunieron páginas de las universidades de Cornell (620), Texas (827), Washington (1 205) y Wisconsin (1 263), además de $4\,120$ páginas de otras universidades.

2.2.1. Preproceso y partición

En el preproceso se han eliminado las cabeceras MIME, las marcas html (de la forma <...>) y las palabras que aparecen únicamente una vez. No se han eliminado las llamadas *stopwords*, pero sí se han *tokenizado* los números.

Se aconseja no utilizar las páginas de una misma universidad para training y para test, por lo tanto se han realizado cuatro particiones distintas en las que el conjunto de test lo forman las páginas pertenecientes a una sola universidad y el resto componen el conjunto de training. Además para poder comparar los resultados obtenidos con otros anteriores sólo se utilizan las cuatro clases ("definidas", es decir, sin incluir la categoría "otros") con más páginas, es decir, student, faculty, course y project. La longitud media de los documentos utilizados es de 259.5 palabras.

alt.atheism	comp.graphics	comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x
misc.forsale	rec.autos	rec.motorcycles
rec.sport.baseball	rec.sport.hockey	sci.crypt
sci.electronics	sci.med	sci.space
soc.religion.christian	talk.politics.guns	talk.politics.mideast
talk.politics.misc	talk.religion.misc	

TABLA 2.2: Grupos de noticias del corpus 20 Newsgroups.

2.2.2. Resultados anteriores

Nigam et al. ([NLM99]) consiguen una tasa de error del $7,92\,\%$ utilizando una técnica de máxima entropía y selección de vocabulario. Utilizando un clasificador de Bayes, más similar a nuestra aproximación obtienen una tasa de error del $13,69\,\%$, también utilizando selección del vocabulario. En la página web del grupo ([Gro]) se pueden encontrar numerosos resultados de distintas técnicas aplicada a este corpus de datos.

2.3. 20 Newsgroups

El corpus "20 Newsgroups", como sugiere el nombre, está compuesto de mensajes mandados a 20 grupos de noticias, un listado de los cuales aparece en la tabla 2.2. Contiene un total de aproximadamente $20\,000$ mensajes distribuidos de forma casi uniforme en todos los grupos. Se puede observar que la distinción entre algunos grupos es ciertamente borrosa, ya que hay por ejemplo distintos grupos dedicados a religión (o ateísmo) y a ordenadores.

2.3.1. Preproceso y partición

Como preproceso del corpus se han eliminado las cabeceras[1] y las secciones codificadas correspondientes a archivos binarios (codificación UU) y se han eliminado las *stopwords*. La longitud media resultante es de 127.6 palabras por documento. Para la partición, a fin de intentar minimizar la inclusión de documentos del conjunto de test en el conjunto de entrenamiento debido a contestaciones en las que se cita el documento original y para intentar simular la situación real en la que se implantaría un clasificador para esta tarea, se han seleccionado los primeros 800 documentos por orden cronológico de cada categoría, obteniendo entonces un conjunto de $16\,000$ documentos de entrenamiento y $3\,974$ de test.

[1] Aunque no las líneas correspondientes a los campos `From:` y `Subject:`.

2.3.2. Resultados anteriores

Alfons Juan y Herrman Ney en [JN02] consiguen una mejor tasa de error del $14.3\,\%$ utilizando un clasificador multinomial. En el capítulo 3 repetiremos parte de los resultados obtenidos en este artículo.

2.4. Nota sobre el preproceso

El formato original de los documentos de todos los corpus es el "formato natural", es decir, los textos como fueron escritos originariamente. La representación que usaremos para trabajar con ellos será sin embargo el de *vectores de cuentas* (ver cap. 3). El preproceso y la transformación al formato adecuado se han llevado a cabo utilizando el programa `rainbow` [McC96].

En múltiples trabajos orientados a clasificación de documentos se aplica una *selección del vocabulario*, es decir, la eliminación de un subconjunto de las palabras que componen el vocabulario completo de la tarea, considerando que la información que aportan no es significativa para un correcta clasificación. Un ejemplo ya lo hemos visto en el corpus de EuTrans y de WebKB al eliminar las palabra con frecuencia de aparición igual a 1 y las stopwords. El objetivo es disminuir la cantidad de parámetros a estimar, lo que a su vez repercute en una reducción de la cantidad de muestras de entrenamiento necesarias para estimarlos de forma correcta. Una técnica habitual para elegir qué palabras se seleccionan es utilizar el concepto de *ganancia de información* (information gain), una heurística basado en la teoría de información que otorga un valor numérico a cada palabra en base a su poder discriminativo (ver por ejemplo [YP97]).

A pesar de que el empleo de estas técnicas en muchos casos mejora los resultados hemos decidido no emplearlos en este proyecto. Una de las razones que nos llevan a esta decisión es que todavía no hay una explicación teórica satisfactoria que justifique la aplicación de estos métodos dentro del contexto de aprendizaje por máxima verosimilitud. Además en el capítulo 6 estudiaremos el aprendizaje no supervisado, es decir, el aprendizaje en base a un conjunto de muestras sin etiqueta de clase, por lo que en este caso nos resultaría imposible aplicar estos métodos.

CAPÍTULO 3

Clasificador multinomial

En este capítulo se presenta el clasificador base con el que vamos a trabajar. Los resultados teóricos presentados, así como la experimentación con el corpus 20 newsgroups se pueden encontrar en [JN02].

3.1. Distribución multinomial

Sea una población de D tipos de elementos en proporciones dadas por el vector de reales $\boldsymbol{p} = (p_1, p_2, \ldots, p_D)$, con $0 \leq p_d \leq 1, \forall 1 \leq d \leq D$ y $\sum_{d=1}^{D} p_d = 1$. Sea a su vez $X_d, 1 \leq d \leq D$ el número de elementos del tipo d obtenidos tras seleccionar x_+ elementos de forma aleatoria, independiente y con reemplazamiento. El vector $\boldsymbol{X} = (X_1, \ldots, X_D)$ es una variable aleatoria *multinomial* de parámetros x_+ y $\boldsymbol{p}$ (en símbolos $\boldsymbol{X} \sim Mult(x_+, \boldsymbol{p})$). Existen D^{x_+} secuencias posibles al seleccionar x_+ elementos aleatoriamente y a cada una de ellas le podemos asociar un vector de contadores $\boldsymbol{x} \in \{0, 1, \ldots, x_+\}^D$ tal que $\sum_{d=1}^{D} x_d = x_+$. La probabilidad de cada una de estas secuencias es $\prod_{d=1}^{D} p_d^{x_d}$, sin embargo la correspondencia entre secuencias y vectores de contadores no es biunívoca, existiendo

$$\binom{x_+}{\boldsymbol{x}} = \frac{x_+!}{x_1! \ldots x_D!} \tag{3.1}$$

secuencias distintas asociadas al vector $\boldsymbol{x}$. De ahí que la función de probabilidad de la variable aleatoria X sea

$$p(\boldsymbol{x}) = \frac{x_+!}{x_1! \ldots x_d!} \prod_{d=1}^{D} p_d^{x_d} \tag{3.2}$$

para todo $\boldsymbol{x} \in \{0, 1, \ldots, x_+\}^D$ tal que $\sum_{d=1}^{D} x_d = x_+$.

3.2. Representación de los documentos

En nuestro caso consideramos como población de origen para cada tarea el vocabulario de los textos (posiblemente restringido) y cada documento vendrá representado por un vector de cuentas de las palabras presentes en el documento. Consideraremos entonces que la distribución de probabilidad condicional $p(\boldsymbol{x}|c)$ es una multinomial, siendo entonces la probabilidad condicional de un documento $\boldsymbol{x} = (x_1, \ldots, x_D)$

$$p(\boldsymbol{x}|c) = \frac{x_+!}{x_1! \ldots x_D!} \prod_d p_{cd}^{x_d} \tag{3.3}$$

La simplicidad de este modelo provoca que se ignoren ciertas características propias de los documentos, como la información que proporciona la secuencialidad del texto, es decir el contexto que rodea cada palabra. Otro punto a notar es que se está asumiendo que la longitud de los documentos está fijada por una constante (la x_+ en la definición de la distribución multinomial), hecho que es indudablemente falso. Podríamos considerar la probabilidad condicional $p(l(\boldsymbol{x})|c)$, donde $l(\boldsymbol{x})$ representa la longitud del documento $\boldsymbol{x}$, quedando entonces la probabilidad conjunta

$$p(\boldsymbol{x}|c) = p(l(\boldsymbol{x})|c)p'(\boldsymbol{x}|c) \tag{3.4}$$

siendo $p'(\boldsymbol{x}|c)$ una distribución multinomial. Sin embargo, dado que

$$p(c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|c)p(c)}{p(\boldsymbol{x})} = \frac{p(l(\boldsymbol{x})|c)p'(\boldsymbol{x}|c)p(c)}{p(\boldsymbol{x})} \tag{3.5}$$

y recordando la regla de clasificación desarrollada en el capítulo 1 (ecuación 1.2, pág 5) obtenemos

$$\begin{aligned} G(\boldsymbol{x}) &= \operatorname*{argmáx}_{c \in \mathcal{C}} \log p(c|\boldsymbol{x}) = \operatorname*{argmáx}_{c \in \mathcal{C}} \left(\log p(c) + \log p(l(\boldsymbol{x})|c) + \log p'(\boldsymbol{x}|c)\right) \\ &= \operatorname*{argmáx}_{c \in \mathcal{C}} \left(\log p(c) + \log p(l(\boldsymbol{x})|c) + \log l(\boldsymbol{x})! - \log \prod_d x_d! + \log \prod_d p_{cd}^{x_d} \right) \\ &= \operatorname*{argmáx}_{c \in \mathcal{C}} \left(\log p(c) + \log p(l(\boldsymbol{x})|c) + \sum_d x_d \log p_{cd} \right) \end{aligned} \tag{3.6}$$

Donde los p_{cd} son los adecuados al modelo para la longitud $l(\boldsymbol{x})$. Este desarrollo muestra que el término $\binom{x_+}{\boldsymbol{x}}$ no influye en la clasificación, limitándose la influencia de la longitud del documento únicamente a la probabilidad condicional $p(l(\boldsymbol{x})|c)$ y a la elección de los p_{cd} correspondientes. Sin embargo, si asumimos que las distintas longitudes de los documentos son equiprobables podemos también ignorar el término $p(l(\boldsymbol{x}|c)$ de la función de clasificación $G(\boldsymbol{x})$ y considerar que los parámetros p_{cd} son iguales para todas las longitudes de los documentos a clasificar, es

decir, que la probabilidad de aparición de las palabras en los documentos es independiente de su longitud, con lo que la función de clasificación queda finalmente

$$G(\boldsymbol{x}) = \underset{c \in \mathcal{C}}{\operatorname{argmáx}} \left(\log p(c) + \sum_d x_d \log p_{cd} \right) \tag{3.7}$$

Esta ecuación muestra a su vez que la funciones discriminantes g_c de las distintas clases son funciones lineales.

3.3. Aprendizaje por máxima verosimilitud

El vector $\boldsymbol{\theta}$ de parámetros a estimar está formado por las probabilidades a priori de las distintas clases y los parámetros que gobiernan las distribuciones multinomiales, es decir

$$\boldsymbol{\theta} = (p(1), \ldots, p(C); \boldsymbol{p}_1, \ldots, \boldsymbol{p}_C) \tag{3.8}$$

donde

$$\forall c \in \mathcal{C} : 0 \leq p(c) \leq 1 \wedge \sum_c p(c) = 1 \tag{3.9}$$

$$\forall c \in \mathcal{C}, \forall 1 \leq d \leq D : 0 \leq p_{cd} \leq 1 \wedge \forall c \in \mathcal{C} : \sum_d p_{cd} = 1 \tag{3.10}$$

Dado un conjunto de entrenamiento $\{(\boldsymbol{x}_n, c_n)\}_{n=1}^N$ la función de log-verosimilitud es

$$\log L(\boldsymbol{\theta}) = \sum_n \left(\log p(c_n) + \log p(\boldsymbol{x}_n | c_n) \right) \tag{3.11}$$

y dado que las probabilidades condicionales son multinomiales

$$\log L(\boldsymbol{\theta}) = \sum_n \left(\log p(c_n) + \log x_+! - \sum_d \log x_{nd}! + \sum_d x_{nd} \log p_{cd} \right) \tag{3.12}$$

Por el criterio de máxima verosimilitud queremos maximizar 3.12 sujeto a 3.9 y 3.10. Definimos la función

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \log L(\boldsymbol{\theta}) + \lambda_0 \left(\sum_c p(c) - 1 \right) + \sum_c \left(\lambda_c \left(\sum_d p_{cd} - 1 \right) \right) \tag{3.13}$$

donde $\boldsymbol{\lambda} = (\lambda_0, \lambda_1, \ldots, \lambda_C)$ es un vector de *multiplicadores de Lagrange*. Se cumple entonces[1]

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \text{ s.a (3.9) y (3.10)}}{\operatorname{argmáx}} L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmáx}} \max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{3.14}$$

[1] En adelante denotaremos con un símbolo ˆ los parámetros maximo-verosímiles (p.ej. $\hat{p}_{cd}$).

Se tiene pues que cumplir

$$\begin{aligned} \left.\frac{\partial \mathcal{L}}{\partial \lambda_0}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = 0 \quad \left.\frac{\partial \mathcal{L}}{\partial p(c)}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = 0 \qquad & \forall 1 \leq c \leq C \\ \left.\frac{\partial \mathcal{L}}{\partial \lambda_c}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = 0 \quad \left.\frac{\partial \mathcal{L}}{\partial p_{cd}}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = 0 \qquad & \forall 1 \leq d \leq D \end{aligned} \tag{3.15}$$

Derivando obtenemos

$$\left.\frac{\partial \mathcal{L}}{\partial \lambda_0}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = \sum_c \hat{p}(c) - 1 = 0 \tag{3.16}$$

y para $c = 1, \ldots, C$

$$\left.\frac{\partial \mathcal{L}}{\partial p(c)}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = \sum_{n:c_n=c} \frac{1}{\hat{p}(c)} - \hat{\lambda}_0 = 0 \quad \Rightarrow \quad \hat{p}(c) = \frac{\sum_{n:c_n=c} 1}{\hat{\lambda}_0} = \frac{N_c}{\hat{\lambda}_0} \tag{3.17}$$

siendo N_c la cantidad de documentos perteneciente a la clase c. Sustituyendo 3.17 en 3.16

$$\sum_c \frac{N_c}{\hat{\lambda}_0} = 1 \quad \Rightarrow \quad \hat{\lambda}_0 = \sum_c N_c = N \tag{3.18}$$

y por tanto

$$\hat{p}(c) = \frac{N_c}{N} \tag{3.19}$$

es decir, obtenemos el resultado intuitivamente satisfactorio de que la probabilidad a priori de la clase c es simplemente la proporción de muestras de la clase c respecto del total. Por otro lado tenemos para $c = 1, \ldots, C$

$$\left.\frac{\partial \mathcal{L}}{\partial \lambda_c}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = \sum_d p_{cd} - 1 = 0 \tag{3.20}$$

y

$$\left.\frac{\partial \mathcal{L}}{\partial p_{cd}}\right|_{\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\lambda}}} = \sum_{n:c_n=c} \frac{x_{nd}}{\hat{p}_{cd}} - \hat{\lambda}_c = 0 \quad \Rightarrow \quad \hat{p}_{cd} = \frac{1}{\hat{\lambda}_c} \sum_{n:c_n=c} x_{nd} \tag{3.21}$$

Sustituyendo 3.21 en 3.20

$$\sum_d \frac{1}{\hat{\lambda}_c} \sum_{n:c_n=c} x_{nd} = 1 \tag{3.22}$$

De ahí

$$\hat{\lambda}_c = \sum_d \sum_{n:c_n=c} x_{nd} \tag{3.23}$$

Para simplificar la notación definimos N_{cd} como el número de ocurrencias de la palabra d en los documentos de la clase c

$$N_{cd} = \sum_{n:c_n=c} x_{nd} \tag{3.24}$$

Y sustituyendo 3.23 en 3.21

$$\hat{p}_{cd} = \frac{N_{cd}}{\sum_{d'} N_{cd'}} \tag{3.25}$$

o expresado en forma vectorial

$$\hat{\boldsymbol{p}}_c = \frac{1}{\sum_{d'} N_{cd'}} \sum_{n:c_n=c} \boldsymbol{x}_n \tag{3.26}$$

En el caso (teórico) de que todos los documentos tuvieran longitud contante x_+ podemos simplificar aún más la ecuación 3.23

$$\hat{\lambda}_c = \sum_{n:c_n=c} \sum_{d} x_{nd} = N_c x_+ \tag{3.27}$$

Y por tanto obtendríamos

$$\hat{p}_{cd} = \frac{N_{cd}}{N_c x_+} \tag{3.28}$$

3.4. Suavizado

Como ya se comentó en el capítulo 1 el aprendizaje por máxima verosimilitud presenta problemas de sobreentrenamiento, lo que puede ocasionar que el clasificador no generalice adecuadamente. Un caso evidente de este problema se muestra en el siguiente ejempo:

> *Supongamos que una palabra d del vocabulario no aparece en ningún documento de la clase c, es decir, $N_{cd} = 0$. Por la ecuación 3.25 tendremos*
>
> $$\hat{p}_{cd} = \frac{N_{cd}}{\sum_{d'} N_{cd'}} = 0$$
>
> *Por tanto, si en la fase de test aparece un documento $\boldsymbol{x}$ pertenciente a la clase c que* sí *contiene la palabra d, la probabilidad a posteriori $p(c|\boldsymbol{x})$ será nula, por lo que lo clasificaremos en una clase errónea.*

Una solución a este problema consiste en alterar de una forma controlada los parámetros estimados. Nótese que con esto alteramos el criterio de aprendizaje que nos habíamos fijado (maximizar la (log-)verosimilitud). En este apartado presentaremos una técnica conocida como *suavizado* en distintas modalidades.

La primera técnica, conocida como *suavizado de Laplace* es la más directa de las aquí presentadas. Consiste simplemente en añadir un $\epsilon > 0$ a todas las cuentas, con lo que obtendríamos

$$\hat{p}_{cd} = \frac{N_{cd} + \epsilon}{\sum_{d'} (N_{cd'} + \epsilon)} \tag{3.29}$$

Está claro que esta aproximación resuelve el problema de las parámetros nulos expuesto en el ejemplo anterior, sin embargo es un método "ciego", en el sentido de que no realiza ninguna distinción entre los distintos tipos de parámetros. Existen otras técnicas de suavizado más elaboradas que han sido estudiadas exhaustivamente en el campo del modelado del lenguaje para el reconocimiento del habla.

Estos métodos se conocen como métodos de *descuento absoluto*. En lugar de añadir cuentas artificiales descontamos cierta masa de probabilidad de los eventos vistos, que después redistribuimos entre un subconjunto de parámetros. La diferencia entre las distintas técnicas radica en qué parámetros reciben esta masa de probabilidad ganada y la distribución que utilizamos para la redistribución.

La primera técnica, conocida como *back-off* distribuye la masa de probabilidad ganada entre los eventos no vistos, siendo entonces

$$\hat{p}_{cd} = \begin{cases} \dfrac{N_{cd} - b}{\sum_{d'} N_{cd'}} & \text{si } N_{cd} > 0 \\ M \dfrac{\beta_d}{\sum_{d':N_{cd'}=0} \beta_{d'}} & \text{si } N_{cd} = 0 \end{cases} \tag{3.30}$$

donde b es un parámetro del suavizado (restringido al intervalo $(0, 1)$ para evitar probabilidades nulas), M es la masa de probabilidad ganada

$$M = \frac{b\,|\{d' : N_{cd'} > 0\}|}{\sum_{d'} N_{cd'}} \tag{3.31}$$

y β_d es una *distribución generalizada*, como por ejemplo la *distribución uniforme*

$$\beta_d = \frac{1}{D} \tag{3.32}$$

o la *distribución unigrama*

$$\beta_d = \frac{\sum_c N_{cd}}{\sum_{d'} \sum_c N_{cd'}} \tag{3.33}$$

El segundo método, conocido como *interpolación*, distribuye la masa de probabilidad ganada entre todos los eventos

$$\hat{p}_{cd} = \text{máx}\left\{0, \frac{N_{cd} - b}{\sum_{d'} N_{cd'}}\right\} + M\beta_d \tag{3.34}$$

donde M sigue siendo la masa de probabilidad ganada, expresada en la ecuación 3.31.

3.5. Experimentación

Se ha realizado la experimentación con los tres corpus presentados en el capítulo 2. Los resultados se muestran en las figuras 3.1, 3.2 y 3.3 y los mejores resultados obtenidos con cada

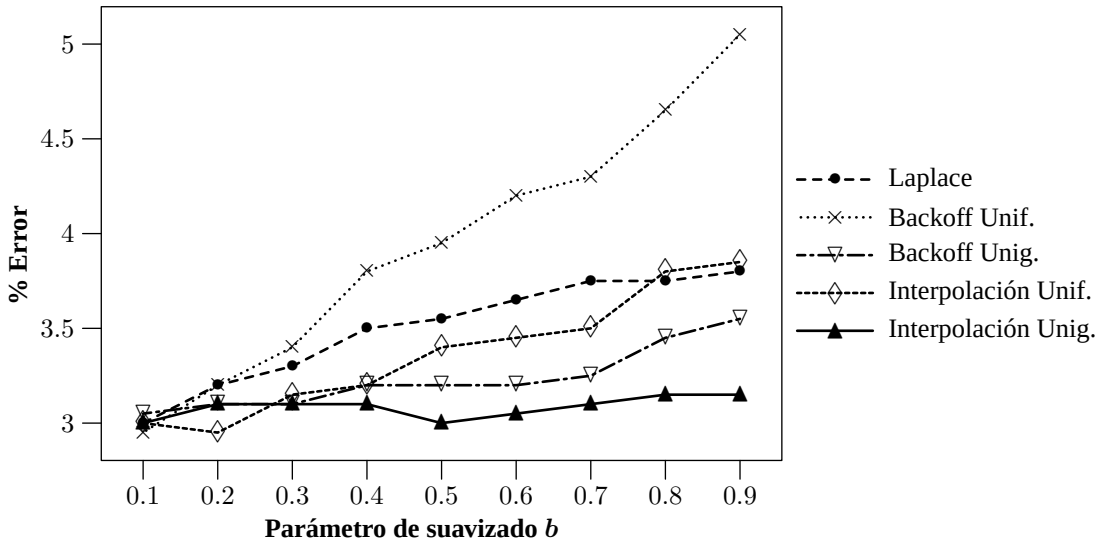

FIGURA 3.1: Tasa de error con el corpus EuTrans para distintos métodos de suavizado.

	EuTrans	**WebKB**	**20news**
Laplace	3.00 %	17.96 %	14.87 %
Backoff Unif.	2.95 %	15.66 %	15.73 %
Backoff Unig.	3.05 %	15.95 %	14.72 %
Interpolación Unif.	2.95 %	17.00 %	15.75 %
Interpolación Unig.	3.00 %	17.87 %	14.90 %

TABLA 3.1: Mejores resultados obtenidos con cada método de suavizado.

método se muestran en la tabla 3.1. Se puede observar que utilizar un tipo de suavizado u otro tiene un efecto significativo sobre la tasa de error. Sin embargo no se puede encontrar ninguna característica que permita decidir qué tipo de suavizado es el más conveniente, dependiendo pues de la tarea y del corpus sobre los que queramos trabajar. Así, por ejemplo, en el corpus EuTrans en general parece más conveniente utilizar valores relativamente bajos para el parámetros de suavizado y el menor error se obtiene utilizando interpolación uniforme. Por contra para el corpus WebKB obtenemos generalmente mejores resultados para valores altos de b, alcanzando el menor error al utilizar backoff uniforme.

La figura 3.4 muestra la matriz de confusión para el corpus 20 newsgroups con el mejor resultado obtenido. Este corpus facilita la interpretación de los resultados, ya que permite ver las relaciones entre las distintas clases de forma intuitiva. Así por ejemplo la matriz muestra que, efectivamente, donde encontramos más errores es en los grupos de temática similar, como los 4 grupos de informática (incluyendo también el grupo de electrónica) o los grupos acerca

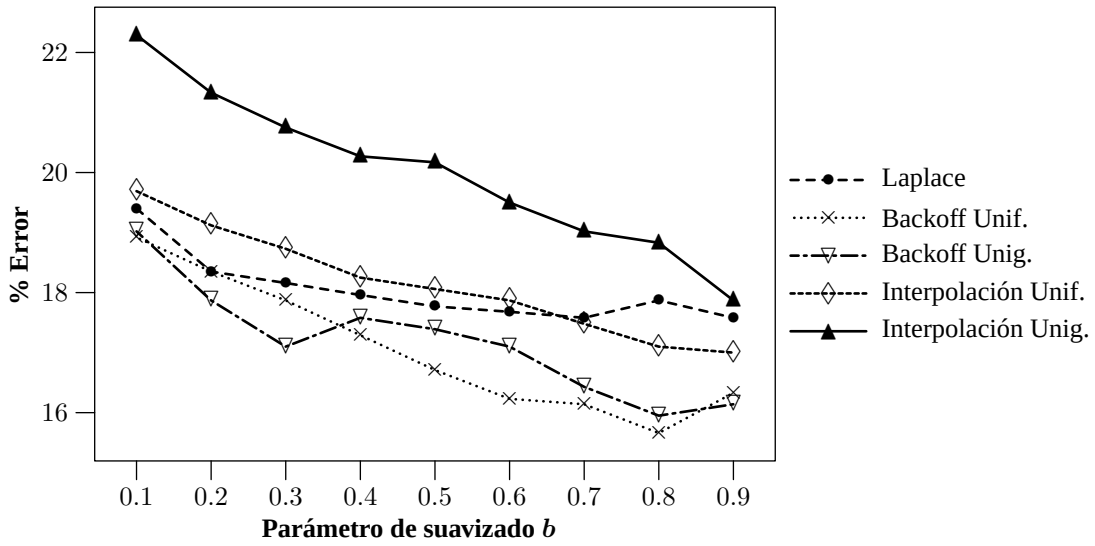

FIGURA 3.2: Tasa de error con el corpus WebKB para distintos métodos de suavizado.

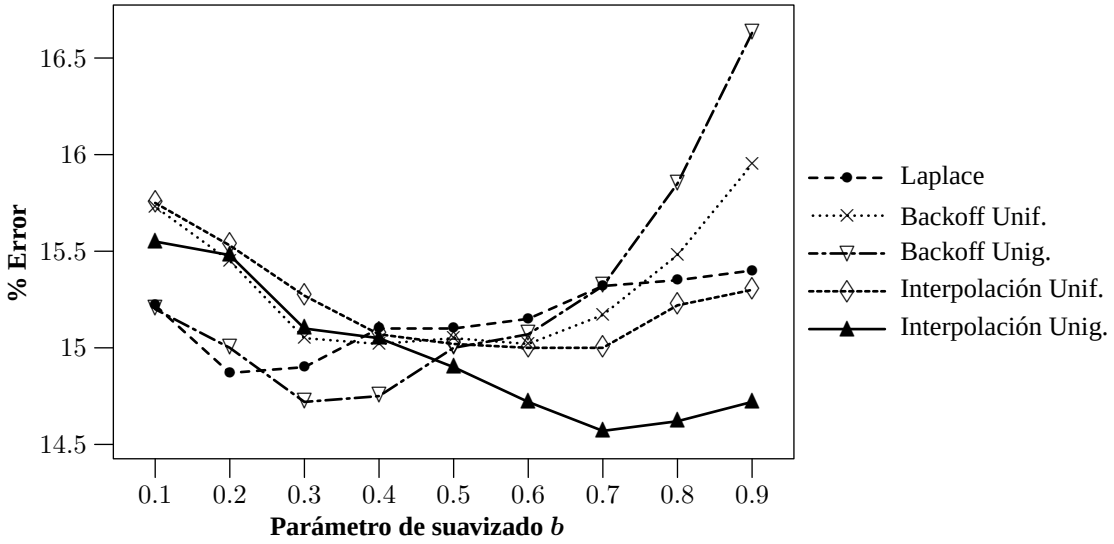

FIGURA 3.3: Tasa de error con el corpus 20 newsgroups para distintos métodos de suavizado.

de política. Este resultado no es sorprendente, ya que en el caso de realizar una clasificación manual, porbablemente la decisión no estaría en absoluta clara y distintas personas clasificarían los mismos documentos en distintas categorías.

En el apéndice D se pueden encontrar resultados adicionales, tales como las matrices de confusión de los corpus EuTrans y WebKB, que no se han incluído aquí para no entorpecer la exposición.

	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	156	0	0	0	0	0	1	0	1	0	0	0	0	0	1	4	0	0	2	35
comp.graphics	0	143	11	0	0	0	5	0	0	0	0	8	6	0	3	0	0	0	0	1
comp.os.ms-windows.misc	0	15	123	17	6	11	4	0	0	1	0	1	1	0	0	0	0	0	2	2
comp.sys.ibm.pc.hardware	0	2	11	158	9	0	7	0	0	1	0	0	11	0	1	0	0	0	0	0
comp.sys.mac.hardware	0	3	3	4	182	0	4	0	0	0	0	0	1	1	0	0	0	0	1	0
comp.windows.x	0	16	11	7	3	156	0	0	2	0	0	0	0	0	2	0	1	0	0	0
misc.forsale	0	0	1	9	4	1	172	3	3	0	1	0	6	0	0	0	0	0	0	0
rec.autos	0	1	1	1	0	0	5	189	1	0	0	0	1	0	0	0	0	0	1	0
rec.motorcycles	0	0	0	0	0	0	2	3	192	0	0	0	3	0	0	0	0	0	0	0
rec.sport.baseball	0	0	0	0	0	0	1	2	0	196	1	0	0	0	0	0	0	0	0	0
rec.sport.hockey	0	0	0	0	0	0	0	0	0	5	191	2	0	0	1	0	0	0	0	0
sci.crypt	0	3	1	0	1	1	4	0	0	0	0	183	2	1	0	0	1	0	1	0
sci.electronics	1	6	1	7	1	1	3	6	2	0	0	1	165	4	1	1	0	0	0	0
sci.med	0	4	0	1	0	0	1	1	0	0	2	0	3	183	2	0	0	0	0	3
sci.space	2	3	0	1	1	0	0	0	0	0	0	2	2	3	182	0	0	0	4	0
soc.religion.christian	1	1	1	2	0	0	0	0	0	0	1	0	1	1	1	183	0	0	1	4
talk.politics.guns	1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	184	0	4	6
talk.politics.mideast	2	1	0	0	0	0	0	0	1	1	0	0	0	1	0	3	0	186	5	0
talk.politics.misc	1	1	1	0	1	0	0	0	0	0	0	0	0	1	2	0	28	12	133	20
talk.religion.misc	42	1	0	0	0	0	0	0	0	0	0	1	0	0	1	8	9	0	6	132

FIGURA 3.4: Matriz de confusión para el corpus 20 newsgroups.

CAPÍTULO 4

Mixturas finitas. El algoritmo EM

En esta capítulo expondremos el modelo matemático en el que se basa el trabajo principal de este proyecto, presentando unas distribuciones de probabilidad conocidas como *mixturas finitas*, así como el algoritmo EM, que nos permitirá estimar los parámetros de estas distribuciones.

4.1. Mixturas finitas

Sea una variable aleatoria X que puede tomar valores dentro de un espacio $\mathcal{X}$ tal que su función de densidad de probabilidad es

$$p(x) = \pi_1 f_1(x) + \cdots + \pi_K f_K(x) \quad \forall x \in \mathcal{X} \tag{4.1}$$

donde

$$\pi_j > 0,\ \forall 1 \leq j \leq K \quad \wedge \quad \sum_{j=1}^{K} \pi_j = 1 \tag{4.2}$$

y

$$f_j(\cdot) \geq 0, \quad \sum_{x \in \mathcal{X}} f_j(x) = 1,\ \forall 1 \leq j \leq K \tag{4.3}$$

(o sustituyendo el sumatorio de 4.3 por una integral en el caso de distribuciones continuas).

En este caso decimos que que X es una mixtura finita. Los parámetros $\pi_1, \ldots \pi_K$ los denominaremos *pesos (de la mixtura)* y las $f_1(\cdot), \ldots, f_K(\cdot)$ *componentes de la mixtura*. Resulta sencillo comprobar que 4.1 es efectivamente una distribución de probabilidad.

Habitualmente las componentes $f_j(\cdot)$ pertenecen a la misma familia de funciones (aunque la definición admite el caso más general). Introduciendo nuevamente vectores de parámetros $\boldsymbol{\theta}_j$ podemos expresar 4.1 como

$$p(x|\boldsymbol{\theta}) = \sum_{j=1}^{K} \pi_j f(x|\boldsymbol{\theta}_j) \tag{4.4}$$

donde $\boldsymbol{\theta}$ incluye tanto los pesos π_j como los vectores $\boldsymbol{\theta}_j$.

Se pueden considerar dos aplicaciones de esta clase de modelos. Dado un conjunto de muestras $\{x_i\}_{i=1}^N$ consideraremos que estamos realizando una aplicación directa si en la población origen de las muestras efectivamente se pueden identificar distintas categorías o fuentes, de tal forma que cada muestra provenga de una de estas categoría. Un ejemplo, derivado del ejemplo de introducción de [DH73] o [DHS01], sería el caso de que quisiéramos distinguir entre dos o más especies de peces en base a ciertas características como el tamaño o el brillo de las escamas. Dado que para cada especie podemos observar individuos de ambos sexos podemos modelizar la distribución de probabilidad de los atributos como una mixtura de dos componentes, donde $f_1(\cdot)$ denota la distribución de probabilidad para un sexo, $f_2(\cdot)$ la distribución para del otro y los pesos π_1 y π_2 la proporción de los distintos sexos.

Una aplicación indirecta, sin embargo, considera una mixtura únicamente como un *instrumento* matemático para modelizar una distribución de probabilidad desconocida y quizá intratable. Un problema que se nos presenta en esta aproximación (y en determinados casos también en la aplicación directa) es que no conocemos el número de componentes de la mixtura, lo que dificulta la estimación de los parámetros.

4.1.1. Identificabilidad

Un problema al que nos enfrentamos al tratar mixturas es el de la *identificabilidad*, es decir, la existencia de una caracterización única de la mixtura (o en un caso más general del modelo) que estamos utilizando.

Sea una familia de mixtura finitas $\mathcal{H}$ definida como

$$\begin{aligned}\mathcal{H} = \Bigg\{ H(x) : H(x) = \sum_{j=1}^{K} \pi_j f(x, \boldsymbol{\theta}_j), \pi_j > 0, \sum_{j=1}^{K} \pi_j = 1, \\ f(\cdot, \boldsymbol{\theta}_j) \text{ distrib. de probabilidad de parámetros } \boldsymbol{\theta}_j \Bigg\}\end{aligned} \tag{4.5}$$

Decimos que esta familia es *identificable* si no existen dos mixturas H_1 y H_2 que sean equivalentes, es decir,

$$\mathcal{H} \text{ identificable} \Leftrightarrow \neg\exists H_1, H_2 \in \mathcal{H}, H_1 \neq H_2 : \forall x \; H_1(x) = H_2(x) \tag{4.6}$$

Naturalmente resulta deseable que las mixturas con las que trabajemos sean identificables, ya que si no no podemos tener garantía alguna de que efectivamente estemos aprendiendo la mixtura correcta, independientemente de la cantidad de muestras de entrenamiento que tengamos.

Existen una serie de "no-identificabilidades triviales" consistentes por ejemplo en permutar los índices de las componentes de una mixtura o dividir una componente en dos o más componentes iguales, distribuyendo la probabilidad a priori original entre ellas. Claramente esta clase de no-identificabilidad no presenta un problema práctico de gran importancia, por lo que por lo general se puede ignorar.

Existen estudios acerca de la identificabilidad de las mixturas (consultar por ejemplo [TSM85]) en los que se demuestra que la familia de mixturas finitas de distribuciones de probabilidad discretas no son identificables, un resultado que no es favorable para nuestros propósitos. Sin embargo en [CPR00] se muestra que esto no representa un problema práctico importante.

4.2. El algoritmo EM

El modelo de mixturas finitas se puede aplicar a nuestro modelo de clases presentado en el capítulo 1. Efectivamente, si consideramos las componentes como las distribuciones de probabilidad condicionales $p(\boldsymbol{x}|c)$ y los pesos como las probabilidades a priori de las clases, obtenemos el equivalente a la ecuación 4.4

$$p(\boldsymbol{x}) = \sum_c p(c)p(\boldsymbol{x}|c) \tag{4.7}$$

Consideremos pues que disponemos de un conjunto de muestras $\{\boldsymbol{x}_n\}_{n=1}^N$ *sin etiqueta de clase*. Procederemos entonces de forma similar al capítulo 3. La verosimilitud respecto a este conjunto de muestras es

$$L(\boldsymbol{\theta}) = \prod_n p(\boldsymbol{x}_n|\boldsymbol{\theta}) \tag{4.8}$$

y la log-verosimilitud

$$\log L(\boldsymbol{\theta}) = \sum_n \log p(\boldsymbol{x}_n|\boldsymbol{\theta}) \tag{4.9}$$

Asumiendo que la función $L(\boldsymbol{\theta})$ es diferenciable podemos escribir

$$\nabla_{\boldsymbol{\theta}_j} \log L(\boldsymbol{\theta}) = \sum_n \frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}_j} \left(\sum_c p(c)p(\boldsymbol{x}|c;\boldsymbol{\theta}_c) \right) \tag{4.10}$$

Asumiendo que los elementos de $\boldsymbol{\theta}_i$ y $\boldsymbol{\theta}_j$ son funcionalmente independientes si $i \neq j$ podemos simplificar la ecuación anterior

$$\nabla_{\boldsymbol{\theta}_j} \log L(\boldsymbol{\theta}) = \sum_n \frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}_j} \Big(p(j)p(\boldsymbol{x}|j;\boldsymbol{\theta}_j) \Big) \tag{4.11}$$

Por otro lado, dado que

$$p(j|\boldsymbol{x}_n;\boldsymbol{\theta}) = \frac{p(\boldsymbol{x}_n|j;\boldsymbol{\theta}_j)p(j)}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \tag{4.12}$$

tenemos

$$\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} = \frac{p(j|\boldsymbol{x}_n;\boldsymbol{\theta})}{p(\boldsymbol{x}_n|j;\boldsymbol{\theta}_n)p(j)} \tag{4.13}$$

y sustituyendo en 4.11 obtenemos

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_j} \log L(\boldsymbol{\theta}) &= \sum_n \frac{p(j|\boldsymbol{x}_n;\boldsymbol{\theta})}{p(\boldsymbol{x}_n|j;\boldsymbol{\theta}_n)p(j)} \nabla_{\boldsymbol{\theta}_j} \left(p(j)p(\boldsymbol{x}|j;\boldsymbol{\theta}_j)\right) \\ &= \sum_n p(j|\boldsymbol{x}_n;\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}_j} \log p(\boldsymbol{x}_n|j;\boldsymbol{\theta}_j)\end{aligned} \tag{4.14}$$

Siguiendo el criterio de máxima verosimilitud exigiremos por tanto que

$$\nabla_{\boldsymbol{\theta}_j} \log L(\boldsymbol{\theta})\big|_{\hat{\boldsymbol{\theta}}} = \mathbf{0} \tag{4.15}$$

(siendo $\mathbf{0}$ el vector nulo).

Consideremos ahora las probabilidades a priori $p(c)$. Utilizando multiplicadores de Lagrange definimos la función

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \lambda) &= \log L(\boldsymbol{\theta}) - \lambda \left(\sum_c p(c) - 1 \right) \\ &= \sum_n \log \sum_c p(c) p(\boldsymbol{x}_n|c;\boldsymbol{\theta}) - \lambda \left(\sum_c p(c) - 1 \right)\end{aligned} \tag{4.16}$$

(recordemos que tenemos que exigir que $\sum_c p(c) = 1$). Derivando

$$\left.\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \lambda)}{\partial \lambda}\right|_{\hat{\boldsymbol{\theta}}, \hat{\lambda}} = -\sum_c \hat{p}(c) + 1 = 0 \tag{4.17}$$

y

$$\begin{aligned}\left.\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \lambda)}{\partial p(c)}\right|_{\hat{\boldsymbol{\theta}}, \hat{\lambda}} &= \sum_n \frac{1}{\sum_{c'} \hat{p}(c') \hat{p}(\boldsymbol{x}_n|c')} \hat{p}(\boldsymbol{x}_n|c) - \hat{\lambda} \\ &= \sum_n \frac{p(\boldsymbol{x}_n|c)}{\hat{p}(\boldsymbol{x}_n)} - \hat{\lambda} = \sum_n \frac{\hat{p}(c|\boldsymbol{x}_n)}{\hat{p}(c)} - \hat{\lambda} \\ &= \frac{1}{\hat{p}(c)} \sum_n \hat{p}(c|\boldsymbol{x}_n) - \hat{\lambda} = 0\end{aligned} \tag{4.18}$$

por tanto

$$\hat{p}(c) = \frac{1}{\hat{\lambda}} \sum_n p(c|\boldsymbol{x}_n) \tag{4.19}$$

Sustituyendo en 4.17

$$-\sum_c \frac{\sum_n p(c|\boldsymbol{x}_n)}{\hat{\lambda}} + 1 = 0 \tag{4.20}$$

De ahí

$$\hat{\lambda} = \sum_c \sum_n \hat{p}(c|\boldsymbol{x}_n) = \sum_n \sum_c \hat{p}(c|\boldsymbol{x}_n) = \sum_n 1 = N \tag{4.21}$$

Y sustituyendo este valor en 4.19

$$\hat{p}(c) = \frac{1}{N} \sum_n \hat{p}(c|\boldsymbol{x}_n) \tag{4.22}$$

Tenemos por tanto que el estimado máximo verosímil es aquel que, para todo $c \in \mathcal{C}$, cumple la ecuación 4.22 y además

$$\sum_n p(c|\boldsymbol{x}_n; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}_c} \log p(\boldsymbol{x}_n|c; \boldsymbol{\theta}_c) \Big|_{\hat{\boldsymbol{\theta}}} = \mathbf{0} \tag{4.23}$$

Por otra parte, por la regla de Bayes tenemos

$$p(c|\boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n|c)p(c)}{\sum_{c'} p(\boldsymbol{x}_n|c')p(c')} \tag{4.24}$$

es decir, tenemos una serie de ecuaciones para calcular el estimador máximo-verosímil que son mutuamente dependientes. El algoritmo EM ("expectation maximization") intenta encontrar este estimador utilizando el método conocido como ascenso por gradiente ("hill-climbing").

El algoritmo se compone de dos pasos, en el paso E ("estimation") estimamos las probabilidades a posteriori $p(c|\boldsymbol{x}_n)$ mediante la ecuación 4.24. Seguidamente, en el paso M ("maximization"), se calculan los parámetros que maximizan la verosimilitud para estos valores mediante las ecuaciones 4.22 y 4.23. Al modificar estos parámetros, sin embargo, generalmente cambiarán los valores calculados para las probabilidades a posteriori, con lo que volvemos a iterar hasta alcanzar un valor estable de la (log-)verosimilitud. Un esquema del algoritmo se muestra en la figura 4.1.

Cabe señalar que el algoritmo aquí expuesto se puede considerar como un caso particular de un algoritmo más general para obtener el estimador máximo-verosímil para un conjunto de datos, en el que no tenemos un conocimiento completo de las características de todas las muestras (es decir, la representación (vectorial) $\boldsymbol{x}_n$ de los objetos no es completa). En nuestro caso podemos considerar la etiqueta de clase como un característica más, que es desconocida para todas las muestras. Para más detalles[1] consultar [DHS01].

[1] El algoritmo aquí expuesto también es tratado tanto en [DH73] como [DHS01], en el capítulo dedicado a aprendizaje no supervisado o clustering, aunque no con la denominación de algoritmo EM.

algoritmo EM
mientras aumenta $\log L(\boldsymbol{\theta})$ **hacer**
// ***Paso E***
$$\hat{p}(c|\boldsymbol{x}_n) = \frac{\hat{p}(\boldsymbol{x}_n|c)\hat{p}(c)}{\sum_{c'} \hat{p}(\boldsymbol{x}_n|c')\hat{p}(c')}$$
// ***Paso M***
Calcular $\hat{\boldsymbol{\theta}}$ tal que
$$\hat{p}(c) = \frac{1}{N}\sum_n \hat{p}(c|\boldsymbol{x}_n)$$
$$\sum_n p(c|\boldsymbol{x}_n;\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}_c} \log p(\boldsymbol{x}_n|c;\boldsymbol{\theta}_c)\Big|_{\hat{\boldsymbol{\theta}}} = \mathbf{0}$$
fmientras
falgoritmo

FIGURA 4.1: Esqueleto del algoritmo EM

4.2.1. Inicialización

El EM, al tratarse de un algoritmo de ascenso por gradiente, presenta los problemas propios de esta clase de algoritmos. En particular no está garantizada la convergencia a un máximo global, por lo que generalmente sólo podemos garantizar que hemos encontrado una solución subóptima[2]. Un aspecto que influye decisivamente en el comportamiento del algoritmo es la inicialización de los parámetros.

El algoritmo EM admite dos inicializaciones, una correspondiente al paso E, inicializando los valores para las probabilidades a posteriori y otra correspondiente al paso M, inicializando los parámetros de los modelos a estimar. Aunque ambas son correctas, intuitivamente parece más natural utilizar esta última.

En este caso existen varias alternativas. La primera y más sencilla consiste en inicializar los parámetros aleatoriamente. Sin embargo esta técnica provoca naturalmente una gran variabilidad en el comportamiento del algoritmo y difícilmente podemos asegurar que la inicialización así obtenida sea adecuada. Otra opción consiste en seleccionar muestras aleatoriamente y considerarlas como prototipos iniciales. Particularizando a nuestro caso en el que los modelos son distribuciones multinomiales y los objetos documentos, podemos inicializar los parámetros p_{cd} (para una c fijada) mediante el documento $\boldsymbol{x}_n$ mediante

$$p_{cd} = \frac{x_{nd} + \epsilon}{l(\boldsymbol{x}_n) + D \cdot \epsilon} \tag{4.25}$$

es decir, normalizar las cuentas del documento $\boldsymbol{x}_n$ respecto a la longitud total del documento, siendo $\epsilon \geq 0$ una constante para evitar obtener probabilidades nulas. Con este método, aunque

[2]O en ocasiones ni siquiera eso, en caso de que el algoritmo se detenga en un "punto de silla".

algoritmo maxmin
entrada P, d, k
salida $Q \subseteq P$
    $Q = \varnothing; D = (\infty)^P; q = \text{arbitrario}(P);$
    para $t = 1$ **hasta** k **hacer**
        $Q = Q \cup \{q\}; maxmin = 0;$
        para todo $p \in P - Q$ **hacer**
            $dpq = d(p, q);$
            si $dpq < D[p]$ **entonces** $D[p] = dpq$ **fsi**
            si $D[p] > maxmin$ **entonces**
                $q = p;$
                $maxmin = D[p]$
            fsi
        fpara
    fpara
falgortimo

FIGURA 4.2: Algoritmo maxmin

presenta un mejor comportamiento debido a que los parámetros se ajustan a muestras presentes en el conjunto de entrenamiento, seguimos teniendo una alta variabilidad ya que la elección de los objetos se continúa realizando de forma aleatoria.

Una tercera opción, basada en la anterior consiste en sustituir la elección aleatoria de los prototipos por una selección guiada, utilizando, p.ej. el algoritmo maxmin. Este algoritmo, dado un conjunto P de puntos en un espacio y una métrica d devuelve un subconjunto de k de esos puntos, elegidos siguiendo una estrategia voraz, seleccionando en la etapa t aquel punto tal que la distancia al prototipo más cercano de los $t-1$ elegidos anteriormente sea máxima. Más formalmente, denominando Q_t al subconjunto de puntos elegidos en la etapa t, $1 \leq t \leq k$

$$Q_t = \begin{cases} \text{arbitrario}(P), & \text{si } t = 1 \\ Q_{t-1} \cup \{q_t\}, & \text{si } t > 1 \end{cases} \tag{4.26}$$

donde

$$q_t = \underset{p \in P - Q_{t-1}}{\text{argmáx}} \; \underset{q \in Q_{t-1}}{\text{mín}} \; d(p, q) \tag{4.27}$$

El esquema del algoritmo se presenta en la figura 4.2. Para facilitar el cálculo del elemento q_t en cada iteración se ha hecho uso de un vector auxiliar D. Para más detalles acerca de éste y otros métodos de inicialización se puede consultar [Jua00].

4.3. Aplicación al caso de multinomiales

Hasta ahora la discusión se ha centrado en el caso general, sin asumir ninguna forma especial para la distribución condicional $p(\boldsymbol{x}|c)$. En esta sección particularizaremos la ecuación 4.23 (paso M) para el caso en el que las probabilidades condicionales estén gobernadas por una distribución multinomial. Recordemos que el vector de parámetros $\boldsymbol{\theta}$ es

$$\boldsymbol{\theta} = (p(1), \ldots, p(C); \boldsymbol{p}_1, \ldots, \boldsymbol{p}_C) \qquad \text{[3.8, pag. 17]}$$

sujeto a

$$\forall c \in \mathcal{C} : 0 \leq p(c) \leq 1 \wedge \sum_c p(c) = 1 \qquad \text{[3.9, pag. 17]}$$

$$\forall c \in \mathcal{C}, \forall 1 \leq d \leq D : 0 \leq p_{cd} \leq 1 \wedge \forall c \in \mathcal{C} : \sum_d p_{cd} = 1 \qquad \text{[3.10, pag. 17]}$$

Las probabilidades a priori ya las hemos calculado (ec. 4.22). Para calcular los p_{cd} definimos nuevamente una función utilizando multiplicadores de Lagrange

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \log L(\boldsymbol{\theta}) - \sum_c \left(\lambda_c \left(\sum_d p_{cd} - 1 \right) \right) \tag{4.28}$$

Siendo $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_C)$. Derivando

$$\left. \frac{\partial \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda})}{\partial \lambda_c} \right|_{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\lambda}}} = -\sum_d \hat{p}_{cd} + 1 = 0 \tag{4.29}$$

Por otra parte, por la ecuación 4.14 sabemos que

$$\frac{\partial \log L(\boldsymbol{\theta})}{\partial p_{cd}} = \sum_n p(c|\boldsymbol{x}_n) \frac{\partial}{\partial p_{cd}} \log p(\boldsymbol{x}_n|c) \tag{4.30}$$

Centrándonos en esta derivada

$$\begin{aligned} \frac{\partial}{\partial p_{cd}} \log p(\boldsymbol{x}_n|c) &= \frac{\partial}{\partial p_{cd}} \log \left(\frac{x_+!}{\prod_{d'} x_{nd'}!} \prod_{d'} p_{cd'}^{\boldsymbol{x}_n d'} \right) \\ &= \frac{\partial}{\partial p_{cd}} \left(\log x_+! - \sum_{d'} \log x_{nd'}! + \sum_{d'} x_{nd'} \log p_{cd'} \right) \\ &= \frac{x_{nd}}{p_{cd}} \end{aligned} \tag{4.31}$$

Por tanto tenemos

$$\begin{aligned}\left.\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda})}{\partial p_{cd}}\right|_{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\lambda}}} &= \left.\frac{\partial \log L(\boldsymbol{\theta})}{\partial p_{cd}}\right|_{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\lambda}}} - \left.\frac{\partial}{\partial p_{cd}}\left(\sum_{c'}\left(\lambda_c\left(\sum_d p_{cd} - 1\right)\right)\right)\right|_{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\lambda}}} \\ &= \sum_n p(c|\boldsymbol{x}_n)\frac{x_{nd}}{\hat{p}_{cd}} - \hat{\lambda}_c\end{aligned} \tag{4.32}$$

y de ahí

$$\hat{p}_{cd} = \frac{1}{\hat{\lambda}_c}\sum_n p(c|\boldsymbol{x}_n)x_{nd} \tag{4.33}$$

Sustituyendo en 4.29

$$-\sum_d \frac{\sum_n x_{nd}\,\hat{p}(c|\boldsymbol{x}_n)}{\hat{\lambda}_c} + 1 = 0 \tag{4.34}$$

Despejando λ_c obtenemos

$$\hat{\lambda}_c = \sum_d \sum_n x_{nd}\,\hat{p}(c|\boldsymbol{x}_n) \tag{4.35}$$

que sustituimos en 4.33 para obtener

$$\hat{p}_{cd} = \frac{1}{\sum_n \sum_d x_{nd}\,\hat{p}(c|\boldsymbol{x}_n)}\sum_n \hat{p}(c|\boldsymbol{x}_n)x_{nd} \tag{4.36}$$

o expresado en forma vectorial

$$\boldsymbol{p}_c = \frac{1}{\sum_n \sum_d x_{nd}\,\hat{p}(c|\boldsymbol{x}_n)}\sum_n \hat{p}(c|\boldsymbol{x}_n)\boldsymbol{x}_n \tag{4.37}$$

4.4. Suavizado

Al igual que hicimos en la sección 3.4 en el caso del algoritmo EM también podemos suavizar los parámetros una vez acabado el entrenamiento, a fin de evitar los problemas derivados del sobreentrenamiento. Si observamos la estructura de las ecuaciones que derivamos en ese caso, con las que hemos obtenido para el algoritmo EM, observamos que su estructura es muy similar. En el capítulo 3 definíamos N_{cd} como

$$N_{cd} = \sum_{n:c_n=c} x_{nd} \qquad \text{[3.24, pag. 18]}$$

es decir, la cantidad de apariciones del elemento (palabra) d en los documentos de la clase c. Podemos generalizar esta definición, considerando la probabilidad de pertenencia de un documento a una clase, obteniendo

$$N_{cd} = \sum_n p(c|\boldsymbol{x}_n)x_{nd} \tag{4.38}$$

Observemos que la ecuación 3.24 es un caso particular de 4.38 en el que, al saber la pertenencia de una muestra a una clase, las probabilidades $p(c|\boldsymbol{x}_n)$ son 0 o 1. Utilizando este término la ecuación 4.36 que define la estimación de los parámetros p_{cd} en el algoritmo EM obtenemos

$$\hat{p}_{cd} = \frac{N_{cd}}{\sum_{d'} N_{cd'}} \tag{4.39}$$

forma totalmente equivalente a la ecuación 3.25. Teniendo esto en cuenta podemos utilizar las mismas ecuaciones de suavizado que desarrollamos para el caso de un clasificador multinomial, adaptándolas al hecho de que ahora los N_{cd} son en general números reales mientras que en el capítulo 3 podíamos asegurar que eran números enteros. Esta adaptación es directa y así, por ejemplo para el caso del suavizado back-off (ec. 3.30, pag. 20) obtenemos

$$\hat{p}_{cd} = \begin{cases} \dfrac{N_{cd} - b}{\sum_{d'} N_{cd'}} & \text{si } N_{cd} > b \\ M \dfrac{\beta_d}{\sum_{d':N_{cd'} \leq b} \beta_{d'}} & \text{si } N_{cd} \leq 0 \end{cases} \tag{4.40}$$

con

$$M = \frac{\sum_{d'} \text{mín}(b, N_{cd'})}{\sum_{d'} N_{cd'}} \tag{4.41}$$

4.5. Cálculo robusto del paso E

Hemos visto que en el paso E necesitamos calcular las probabilidades a posteriori $p(c|\boldsymbol{x}_n)$. Aunque la forma de calcularlas está perfectamente definida por la ecuación 4.24, dado que trabajamos tanto con número muy pequeños (en el sumatorio $\sum_d p_{cd}^{x_{nd}}$) como con números muy grandes (para calcular el término multinomial $\binom{x_+}{\boldsymbol{x}_n}$ tenemos que calcular una serie de factoriales) tenemos problemas numéricos que impiden la aplicación directa de esta ecuación. Recordemos que en el capítulo 3 resolvíamos problemas similares utilizando funciones discriminantes *equivalentes* a la hora de clasificar, pero en este caso sí necesitamos conocer el valor exacto de la probabilidad a posteriori. Afortunadamente, con una serie de sencillas transformaciones

podemos nuevamente resolver esta dificultad, como muestran la siguiente ecuación

$$\begin{aligned} p(c|\boldsymbol{x}_n) &= \frac{p(\boldsymbol{x}_n|c)p(c)}{\sum_{c'} p(\boldsymbol{x}|c')p(c')} = \frac{\left(\binom{x_+}{\boldsymbol{x}_n} \prod_d p_{cd}^{x_{nd}}\right) p(c)}{\sum_{c'} \left(\binom{x_+}{\boldsymbol{x}_n} \prod_d p_{c'd}^{x_{nd}}\right) p(c')} \\ &= \frac{p(c) \prod_d p_{cd}^{x_{nd}}}{\sum_{c'} p(c') \prod_d p_{c'd}^{x_{nd}}} = \frac{(p(c) \prod_d p_{cd}^{x_{nd}}) \Big/ \text{máx}_{c''} \left(p(c'') \prod_d p_{c''d}^{x_{nd}}\right)}{\sum_{c'} (p(c') \prod_d p_{c'd}^{x_{nd}}) \Big/ \text{máx}_{c''} \left(p(c'') \prod_d p_{c''d}^{x_{nd}}\right)} \\ &= \frac{\exp\left(\log p(c) + \sum_d x_{nd} \log p_{cd} - \text{máx}_{c''} \left(\log p(c'') + \sum_d x_{nd} \log p_{c''d}\right)\right)}{\sum_{c'} \exp\left(\log p(c') + \sum_d x_{nd} \log p_{c'd} - \text{máx}_{c''} \left(\log p_{c''d} + \sum_d x_{nd} \log p(c'')\right)\right)} \end{aligned} \tag{4.42}$$

es decir, podemos suprimir el cálculo del término multinomial y nuevamente aplicamos logaritmos al cálculo de las probabilidades.

4.6. Cálculo de la log-verosimilitud

Si desarrollamos el término que nos da la log-verosimilitud

$$\begin{aligned} \log L(\boldsymbol{\theta}) &= \log \prod_n p(\boldsymbol{x}) = \sum_n \log \sum_c p(c) p(\boldsymbol{x}_n|c) \\ &= \sum_n \log \sum_c \left(p(c) \binom{x_+}{\boldsymbol{x}_n} \prod_d p_{cd}^{x_d} \right) \end{aligned} \tag{4.43}$$

podemos observar que para determinar si se incrementa o si por lo contrario permanece constante, es posible ignorar los términos multinomiales, ya que son constantes en cada iteración, pudiendo escribir por tanto

$$\log L(\boldsymbol{\theta}) \propto \sum_n \log \sum_c \left(p(c) \prod_d p_{cd}^{x_d} \right) \tag{4.44}$$

y aplicando transformaciones similares a las que usamos en el cálculo del paso E

$$\begin{aligned}\log L(\boldsymbol{\theta}) &\propto \sum_n \log \max_{c''} \left(p(c'') \prod_d p_{c''d}^{x_d} \right) \sum_c \frac{(p(c) \prod_d p_{cd}^{x_d})}{\max_{c''} (p(c'') \prod_d p_{c''d}^{x_d})} \\ &= \sum_n \left(\max_{c''} \left(\log p(c'') + \sum_d x_{nd} \log p_{c''d} \right) \right. \\ &\quad + \log \sum_c \exp \left(\log p(c) + \sum_d x_{nd} \log p_{cd} \right. \\ &\quad \left. \left. \left. - \max_{c''} \left(\log p(c'') + \sum_d x_{nd} \log p_{c''d} \right) \right) \right)\end{aligned} \tag{4.45}$$

Si observamos los términos que aparecen en esta ecuación podemos observar que todos intervienen también en el cálculo robusto del paso E dado por la ecuación 4.42. Por lo tanto, (una aproximación a) la log-verosimilitud se puede obtener como subproducto del paso E.

CAPÍTULO 5

Aprendizaje supervisado de mixturas

En este capítulo estudiaremos la aplicación de los conceptos expuestos en el capítulo 4 a la tarea de clasificación de texto. Más concretamente, intentaremos modelizar la función de probabilidad condicional de un documento dada una clase como una mixtura de multinomiales.

5.1. El modelo

Como ya hemos visto, la regla de clasificación de máxima probabilidad a posteriori depende en gran medida de cómo modelamos la probabilidad condicional $p(\boldsymbol{x}_n|c)$, siendo $\boldsymbol{x}_n$ un documento a clasificar y c una clase. En el capítulo 3 considerábamos esta distribución de probabilidad como una multinomial, un modelo relativamente sencillo y de aplicación directa que obtiene buenos resultados. En este capítulo estudiaremos una generalización de este modelo, considerando que los textos asociados a una clase no son producidos por una única distribución multinomial sino por un conjunto de éstas, es decir, por una mixtura tal y como la definimos en el capítulo 4.

Definamos entonces la probabilidad $p(\boldsymbol{x}_n|c)$ como

$$p(\boldsymbol{x}_n|c) = \sum_{i \in I_c} p(i) p(\boldsymbol{x}_n|i) \tag{5.1}$$

donde I_c es un conjunto de índices dependiente de la clase c. En el contexto de la clasificación de documentos, el conjunto I_c se puede interpretar de forma intuitiva como el conjunto de subtópicos que pueden aparecer en el conjunto de documentos de temática general modelizada por

la clase c. Teniendo en cuenta esta interpretación en seguida nos podemos dar cuenta de la dificultad que entraña estimar el (cardinal del) conjunto I_c. De llevar a cabo de forma manual está distinción nos encontraríamos con el problema de la definición de estas subclases, así como la clasificación de cada documento en una de ellas.

Nuestra intención en este capítulo es utilizar el algoritmo EM para realizar una división de cada clase en un número dado de subclases de forma automática. En adelante a estas subclases las llamaremos *clases no supervisadas*, ya que en el corpus de entrenamiento no tenemos esta etiqueta de (sub)clase, en contraposición con las *clases supervisadas* que representan el objetivo de nuestra clasificación[1]. Recordemos, no obstante, que nuestro objetivo final no es la identificación de estos subtópicos, pero que utilizaremos esta división como ayuda a la modelización de cada una de las $\mathcal{C}$ clases originales.

5.2. Actualización de la regla de clasificación

En el capítulo 1 expusimos que nuestra regla de clasificación se basaba en maximizar la probabilidad a posteriori $p(c|\boldsymbol{x}_n)$, lo que era equivalente a maximizar el término $\log p(c) + \log p(\boldsymbol{x}_n|c)$ (ec. 1.6, p. 6). Sustituyendo en este término la ecuación 5.1 obtenemos

$$g_c(\boldsymbol{x}_n) = \log p(c) + \log \sum_i p(i)p(\boldsymbol{x}_n|i) \tag{5.2}$$

donde g_c es la función discriminante de la clase c. Aquí nos encontramos con un problema práctico a la hora de calcular el valor de esta función, ya que nuevamente debemos calcular la probabilidad $p(\boldsymbol{x}_n|i)$, que viene dada por la función multinomial y por tanto podemos encontrarnos con problemas numéricos. Para solucionarlo aplicamos una técnica similar a la que utilizamos en el cálculo rebusto del paso E en la sección 4.5.

$$\begin{aligned} g_c(\boldsymbol{x}_n|c) &= \log p(c) + \log \sum_i p(i)p(\boldsymbol{x}_n|i) \\ &= \log p(c) + \log \max_{i''} p(i'')p(\boldsymbol{x}_n|i'') \sum_i \frac{p(i)p(\boldsymbol{x}_n|i)}{\max_{i''} p(i'')p(\boldsymbol{x}_n|i'')} \\ &= \log p(c) + \max_{i''} \log p(i'')p(\boldsymbol{x}_n|i'') \\ &\quad + \log \sum_i \exp\left(\log p(i)p(\boldsymbol{x}_n|i) - \max_{i''} \log p(i'')p(\boldsymbol{x}_n|i'')\right) \end{aligned} \tag{5.3}$$

[1] En las ecuaciones adoptaremos la convención de denotar a las clases supervisadas mediante la variable c, como venimos haciendo hasta este punto y las clases no supervisadas mediante la variable i.

Y sustituyendo la definición de una distribución multinomial (nuevamente podemos ignorar el término $\binom{x_+}{\boldsymbol{x}_n}$)

$$g_c(\boldsymbol{x}_n) = \log p(c) + \max_{i''} \left(\log p(i'') + \sum_d x_{nd} \log p_{id} \right) + \log \sum_i \exp \left[\log p(i) + \sum_d x_{nd} \log p_{id} - \max_{i''} \left(\log p(i'') + \sum_d x_{nd} \log p_{i''d} \right) \right] \tag{5.4}$$

5.3. Condición de parada del entrenamiento

Ya hemos comentado anteriormente el fenómento del sobreentrenamiento en el contexto de los clasificadores multinomiales. Bajo este término entendemos el hecho de que el modelo se adapta demasiado a los datos de entrenamiento y no generaliza de forma satisfactoria. En el contexto de las mixturas de multinomiales, al trabajar con un algoritmo iterativo como el EM nos encontramos nuevamente con este problema a la hora de elegir la condición de parada del entrenamiento. La figura 5.1 muestra la evolución de la log-verosimilitud y del error del modelo sobre el conjunto de *test*. En esta gráfica se puede observar que efectivamente la log-verosimiltud aumenta a medida que incrementamos el número de iteraciones, sin embargo el error alcanza un mínimo en relativamente pocas iteraciones y después aumenta de forma signinficativa.

Una forma habitual de contrarrestar este efecto es utilizar un conjunto de *validación* para detener el entrenamiento. Se trata de un conjunto de muestras que no se utilizan para aprender los parámetros del modelo, pero sobre las que se calcula el error de clasificación con un periodo determinado. El entrenamiento se detiene cuando se ha alcanzado el menor error de clasificación sobre este conjunto. En nuestro caso, para no reducir ni el conjunto de entrenamiento, lo que conllevaría una peor estimación de los parámetros, ni el conjunto de test, con lo que los resultados no sería comparables con trabajos anteriores, utilizamos una técnica de validación cruzada para determinar el número de iteraciones en las que debemos parar el entrenamiento. Sobre el conjunto de entrenamiento definimos una partición en cinco partes de aproximadamente el mismo tamaño y utilizamos cuatro de estas partes ($\sim 80\,\%$ de los datos) para entrenar y la partición restante ($\sim 20\,\%$ de los datos) como conjunto de validación para detener el entrenamiento. Una vez determinado el número medio de iteraciones en los que se consigue el menor error utlizamos este valor para detener el aprendizaje de los parámetros utilizando todo el conjunto de entrenamiento.

5.4. Suavizado intermedio

En la sección 4.4 vimos como aplicar las técnicas de suavizado al finalizar el algoritmo EM. Nos podemos plantear también realizar un suavizado “intermedio”, después de cada iteración.

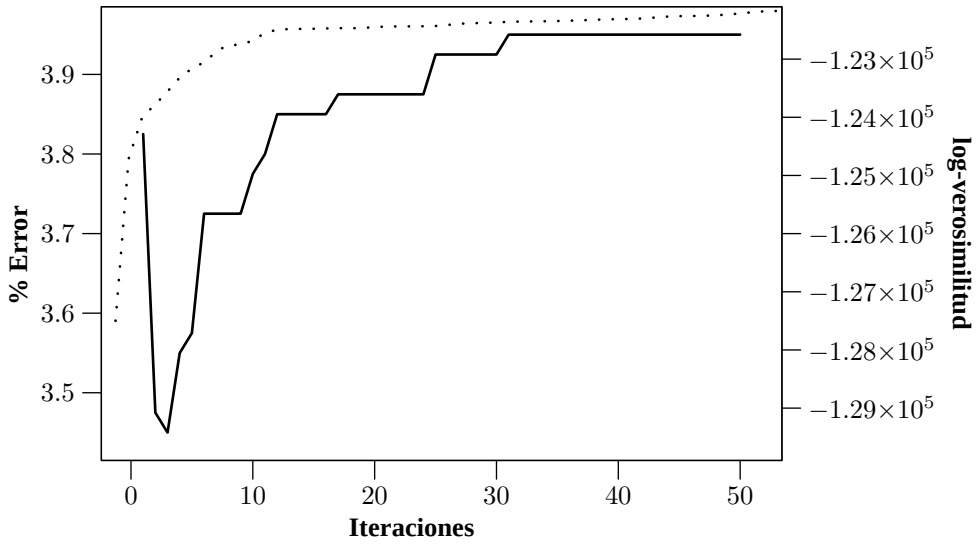

FIGURA 5.1: Evolución del error sobre el conjunto de *test* (trazo continuo) y de la log-verosimilitud (trazo discontinuo) para el algoritmo EM [corpus EuTrans, 2 clases no supervisadas].

Si hacemos esto, estamos perturbando el funcionamiento del algoritmo EM, ya que éste se basa en la maximización de la verosimilitud en cada paso. sin embargo, sí se puede aplicar si garantizamos que tras el suavizado en la iteración i hemos aumentado la verosimilitud respecto a la iteración $i-1$, es decir relajamos el requerimiento de calcular los "mejores" parámetros en cada iteración. La finalidad de esta técnica consiste en intentar perturbar el recorrido del algoritmo por el espacio de parámetros intentando alcanzar un punto con un error menor que el que alcanzaríamos conel funcionamiento inalterado del EM. Sin embargo los resultados experimentales muestran que los resultados obtenidos mediante esta técnica no son superiores a los obtenidos utilizando un suavizado "tradicional" y el coste temporal del algoritmo se ve incrementado, por lo que decidimos no aplicar el suavizado intermedio.

5.5. Experimentación

En esta sección expondremos los resultados experimentales que hemos obtenido para cada uno de los corpus de datos. Un parámetro fundamental para la efectividad del método consiste en la elección adecuada del número de componentes de las mixturas. Lamentablemente no se conoce ningún método para estimar este valor, con lo que debemos determinarlo de forma experimental. Resulta evidente que probar todos los posibles valores (dentro de un rango) para cada una de las clases resulta computacionalmente inviable, por lo que realizamos una simplificación y en cada experimento fijamos el número de componentes de las mixturas, que será igual para cada clase

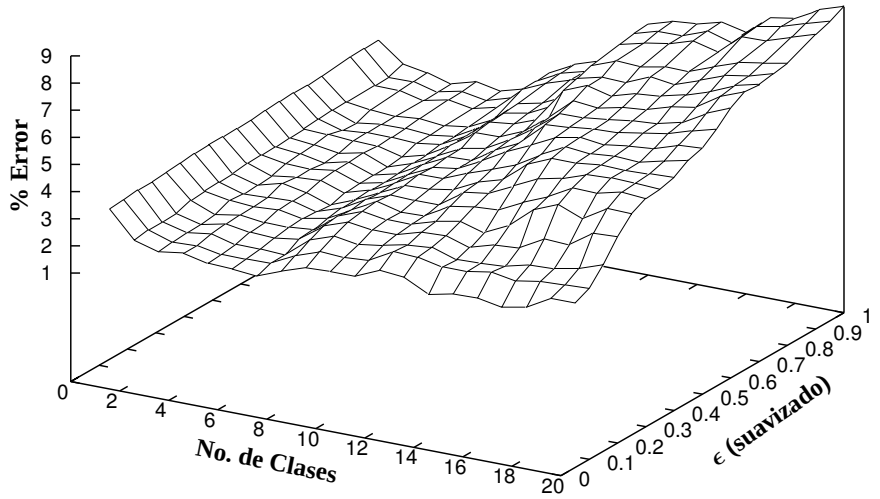

FIGURA 5.2: Tasa de error para el corpus EuTrans, utilizando suavizado de Laplace.

supervisada.

Para disminuir la variabilidad producida por la aleatoriedad de la inicialización (ver sección 4.2.1) se han repetido los experimentos utilizando 5 semillas distintas para la generación de números aleatorios. Los valores aquí expuestos son la media de los resultados obtenidos.

5.5.1. EuTrans

En la figura 5.2 se muestra una representación del error sobre el conjunto de test para el corpus EuTrans utilizando suavizado de Laplace. La imagen sugiere que efectivamente mejoramos la efectividad del clasificador al aumentar el número de componentes de la mixtura. También se puede observar que la curva correspondiente al suavizado de Laplace en la figura 3.1 se corresponde con la sección para $n = 1$ en la figura 5.2.

Para los distintos métodos de suavizado obtenemos un valor distinto del número de clases óptimo a utilizar, como muestra la figura 5.3. Así por ejemplo, para el suavizado de Laplace, el número óptimo de clases es 6 pero para la interpolación uniforme es 7. Resulta intuitivamente reconfortante comprobar que éste es el número medio de subdominios por clase en este corpus, tal como expusimos en la sección 2.1, por lo que podemos asumir que el algoritmo EM efectiva-

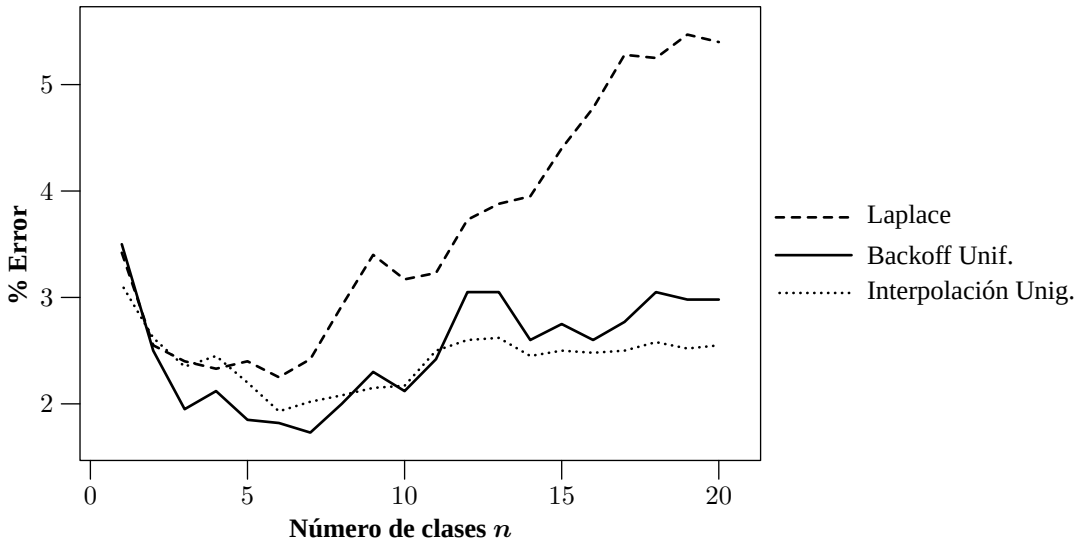

FIGURA 5.3: Tasa de error para el corpus EuTrans con distintos métodos de suavizado variando el número de clases (parám. de suavizado $b = 0.3$).

Suavizado	**Nº de clases**	**Parámetro *b***	**% Error**
Laplace	7	0.05	1.8 %
Backoff Unif.	7	0.3	1.73 %
Backoff Unig.	5	0.9	1.93 %
Interpolación Unif.	7	0.35	1.73 %
Interpolación Unig.	6	0.05	1.9 %

TABLA 5.1: Mejores resultados obtenidos para el corpus EuTrans.

mente ha logrado diferenciar las subclases naturales de esta tarea.

El valor del parámetro b de suavizado también tiene una influencia significativa sobre la efectividad de la clasificación, como muestra la figura 5.4. Un resumen de los mejores resultados obtenidos se muestra en la tabla 5.1.

Se puede observar que esta tarea se beneficia de forma considerable de la utilización de una mixtura de multinomiales respecto al clasificador multinomial sencillo expuesto en el capítulo 3. Esto se puede explicar recordando cómo se generaron estos datos, definiendo cada clase como un conjunto de muestras correspondientes a distintos subdominios, por lo que la aplicación de una mixtura resulta natural. En [JV02] sin embargo se obtienen resultados similares o incluso mejores utilizando una mixtura de Bernoullis, una distribución de probabilidad más simple que la multinomial en el que las muestras son vectores de bits, ignorando por tanto la frecuencia de aparición de cada una de las palabras. El mejor comportamiento de este modelo radica en el hecho

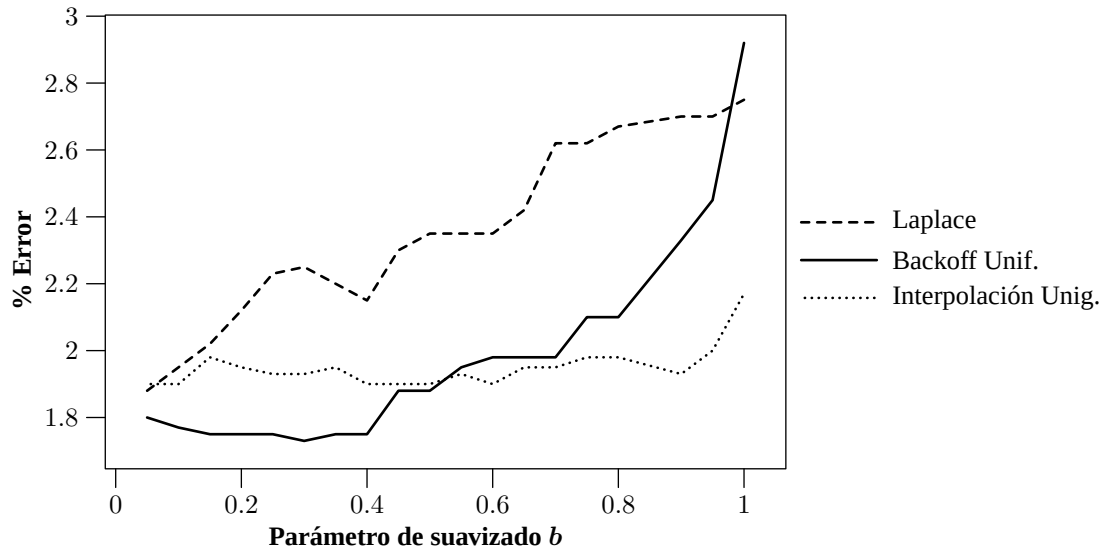

FIGURA 5.4: Tasa de error para distintos métodos de suavizado variando el parámetro b (número de componentes dependiente de cada curva).

de que en este corpus las frases son cortas y apenás existen palabras que lleguen a repetirse en un "documento", por lo que la consideración de las frecuencias no aporta información adicional.

5.5.2. WebKB

La aplicación de una mixtura a la clasificación del corpus WebKB puede resultar más interesante, ya que los datos provienen de una aplicación real y podemos considerar que estamos utilizando el algoritmo EM para encontrar subtópicos dentro de los datos originales.

La figura 5.5 muestra el error sobre el conjunto de test para el corpus WebKB utilizando suavizado de interpolación unigrama. Se observa que este corpus también se beneficia del hecho de utilizar una mixtura de multinomiales como modelo para la probabilidad condicional, pero se observa claramente que utilizar un número demasiado elevado de clases no supervisadas empeora la eficacia del clasificador. Analizando con más detalle los datos, observamos que el mejor resultado lo obtenemos para el suavizado back-off unigrama, con una tasa de error del $14.89\,\%$, es decir un resultado ligeramente superior al obtenido utilizando un clasificador multinomial (ver tabla 3.1, pag. 21).

La tabla 5.2 los mejores resultados obtenidos con las distintas técnicas de suavizado y la figura 5.6 muestra la influencia del número de clases utilizando distintos suavizados (con el parámetro de suavizado óptimo en cada caso). Esta última confirma la conclusión a la que habíamos llegado observando la gráfica 5.5 de que un número demasiado elevado de clases no supervisadas tiene un impacto negativo significativo sobre el clasificador.

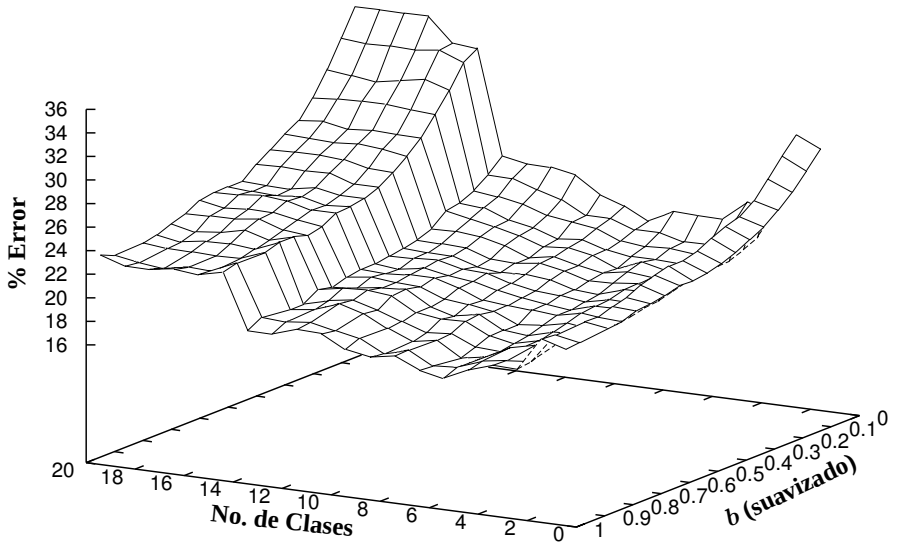

FIGURA 5.5: Tasa de error para el corpus WebKB, utilizando suavizado interpolación unigrama. La orientación de los ejes ha sido cambiada para mejorar la visibilidad de la gráfica.

Suavizado	**Nº de clases**	**Parámetro b**	**% Error**
Laplace	5	0.3	16.43 %
Backoff Unif.	5	0.9	15.56 %
Backoff Unig.	5	0.55	14.89 %
Interpolación Unif.	7	0.5	15.66 %
Interpolación Unig.	6	0.95	16.52 %

TABLA 5.2: Mejores resultados obtenidos para el corpus WebKB.

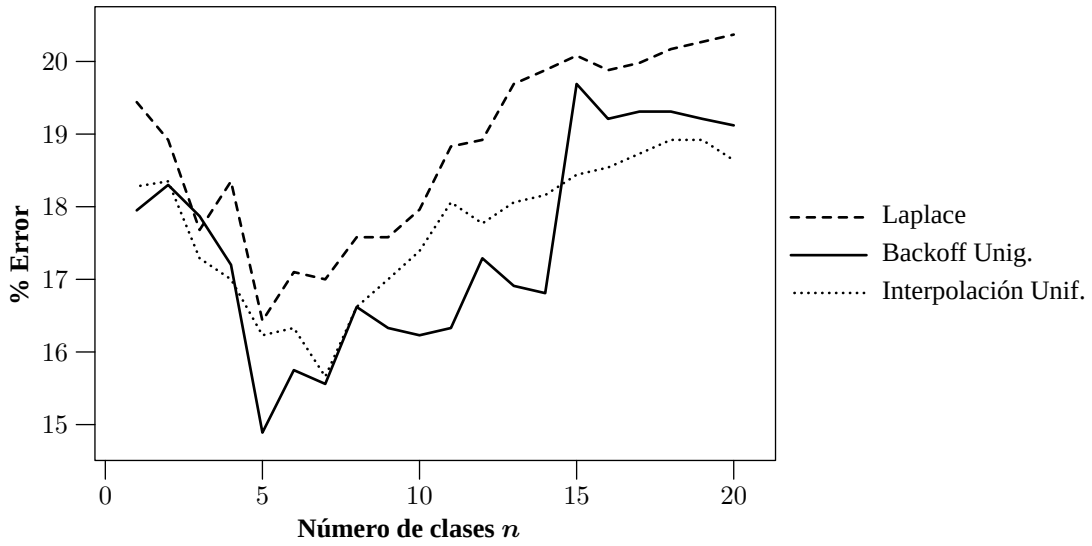

FIGURA 5.6: Tasa de error para el corpus WebKB con distintos métodos de suavizado variando el número de clases.

Suavizado	**Nº de clases**	**Parámetro b**	**% Error**
Laplace	1	0.2	14.87 %
Backoff Unif.	2	0.35	14.75 %
Backoff Unig.	1	0.3	14.67 %
Interpolación Unif.	2	0.3	14.77 %
Interpolación Unig.	1	0.7	14.57 %

TABLA 5.3: Mejores resultados obtenidos para el corpus 20 newsgroups.

5.5.3. 20 newsgroups

Para este corpus, al contrario de lo que sucedía con las tareas anteriores, no obtenemos un beneficio general al utilizar una mixtura. Como se puede observar en la figura 5.7 para el caso particular del suavizado de Laplace, al aumentar el número de clases también aumentamos el error de clasificación. Este comportamiento se observa también para los suavizados back-off e interpolación unigrama. Sí obtenemos una mejora al emplear la variedad uniforme de estos suavizados, que en este caso se comportan de una forma muy similar, como muestra la figura 5.8. Los mejores resultados obtenidos se muestran en la tabla 5.3. Comparando ésta con la tabla 3.1 (pág. 21) comprobamos que el mejor resultado lo obtenemos utilizando un modelo multinomial simple.

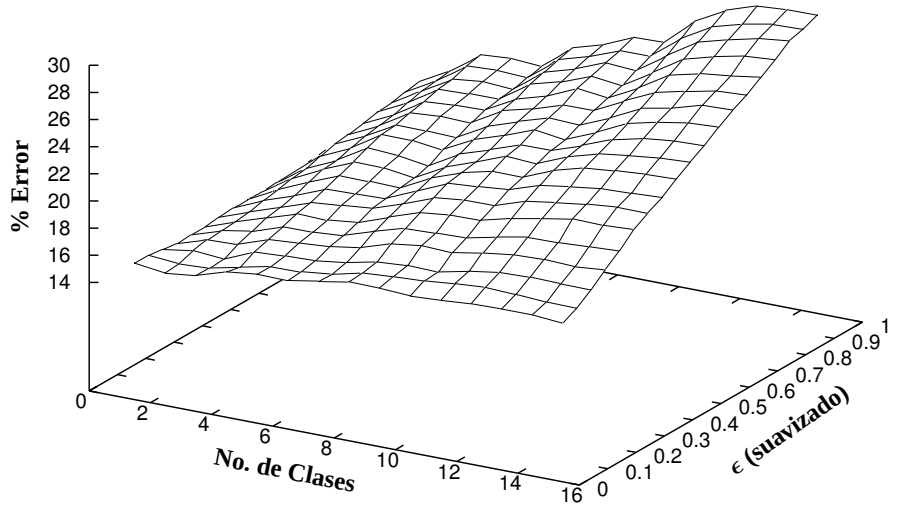

FIGURA 5.7: Tasa de error para el corpus 20 newsgroups, utilizando suavizado de Laplace.

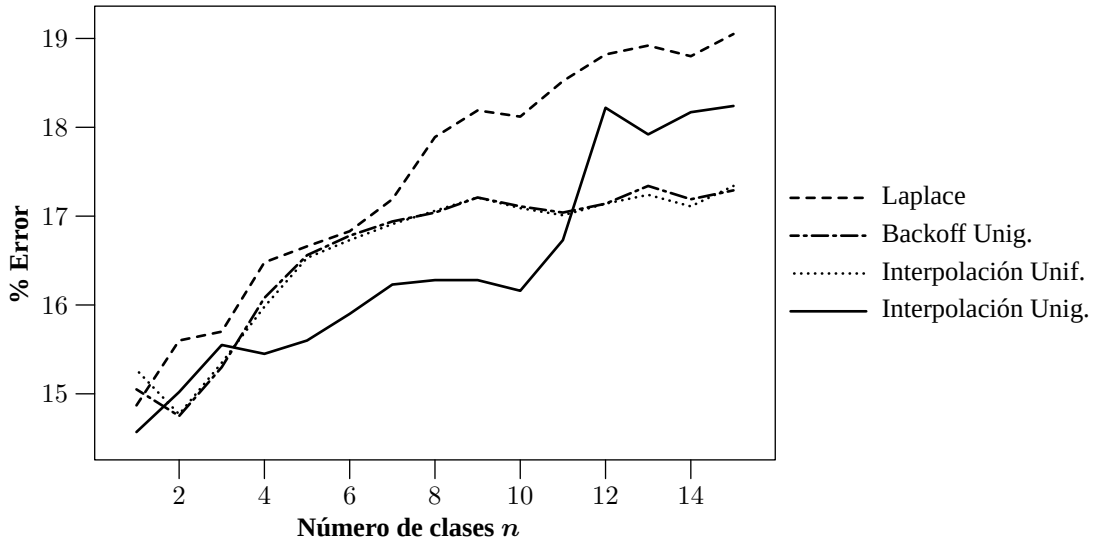

FIGURA 5.8: Tasa de error para el corpus 20 newsgroups con distintos métodos de suavizado variando el número de clases.

CAPÍTULO 6

Aprendizaje no supervisado

En este capítulo nos planteamos la tarea de aprender una mixtura de multinomiales de una forma totalmente no supervisada utilizando el algoritmo EM. Esta aproximación puede resultar especialmente útil en tareas en las que contemos con gran cantidad de documentos sin clasificar y queramos encontrar un etiquetado adecuado de forma automática.

6.1. Aplicación a una tarea de clasificación

Como hemos comentado la aplicación del algoritmo EM de forma totalmente no supervisada es útil para el caso en que no dispongamos de muestras etiquetadas, pero también puede resultar útil en el caso en que tengamos que enfrentarmos a una tarea de clasificación. Como ya se comentó en la sección 5.5 en el caso de modelar cada clase como una mixtura, el número de componentes es el mismo para cada clase supervisada. Naturalmente esto es una restricción que no tiene por qué cumplirse. La aplicación del aprendizaje no supervisado permite que las componentes de la mixtura se distribuyan de forma automática entre las distintas clases supervisadas, con lo que simplificamos el problema de la elección del número de componentes.

También resulta interesante comprobar si el algortimo es capaz de diferenciar las distintas clases supervisadas que hemos definido en el caso de un entrenamiento no supervisado. Esto se lograría si las clases supervisadas estuvieran claramente diferenciadas (respecto al modelo elegido). Entonces el algoritmo EM es capaz de aprender cada una de las componentes de la mixtura. En casos reales esta condición es más difícil que se cumpla y el obviar la información que aporta la etiqueta de clase tiene como resultado que la tarea aumente su complejidad.

Para poder utilizar la mixtura aprendida de forma no supervisada para una tarea de clasificación necesitamos desarrollar las ecuaciones para las funciones discriminantes.

Partimos de la hipótesis de que la clase supervisada c no tiene influencia en la probabilidad condicional para la clase no supervisada i, es decir

$$p(\boldsymbol{x}_n|i,c) = p(\boldsymbol{x}_n|i) \tag{6.1}$$

Para la probabilidad conjunta de una muestra y una clase supervisada tenemos

$$p(\boldsymbol{x}_n,c) = \sum_i p(\boldsymbol{x}_n,c,i) = \sum_i p(\boldsymbol{x}_n,c|i)p(i) \tag{6.2}$$

Para $p(\boldsymbol{x}_n,c|i)$ se cumple

$$p(\boldsymbol{x}_n,c|i) = p(c|i)p(\boldsymbol{x}_n|i,c) \overset{(6.1)}{=} p(c|i)p(\boldsymbol{x}_n|i) \tag{6.3}$$

sustituyendo en 6.2

$$p(\boldsymbol{x}_n,c) = \sum_i p(i)p(c|i)p(\boldsymbol{x}_n|i) \tag{6.4}$$

Dado que

$$\begin{aligned} g_c(\boldsymbol{x}_n) &= \operatorname*{argmáx}_c p(c|\boldsymbol{x}_n) = \operatorname*{argmáx}_c \frac{p(\boldsymbol{x}_n|c)p(c)}{p(\boldsymbol{x}_n)} \\ &= \operatorname*{argmáx}_c p(\boldsymbol{x}_n|c)p(c) = \operatorname*{argmáx}_c p(\boldsymbol{x}_n,c) \end{aligned} \tag{6.5}$$

únicamente nos resta estimar la probabilidad $p(c|i)$. Esta probabilidad se puede definir sencillamente como

$$p(c|i) = \frac{1}{\sum_n p(i|\boldsymbol{x}_n)} \sum_{n:c_n=c} p(i|\boldsymbol{x}_n) \tag{6.6}$$

es decir, la masa de probabilidad asignada a la clase no supervisada i por los documentos de la clase supervisada c respecto a la masa total asociada a la clase no supervisada i.

Nuevamente para evitar problemas de underflow necesitamos realizar una transformación introduciendo la función $\log$ y el operador $\max$, obteniendo

$$\begin{aligned} g_c(\boldsymbol{x}_n) = &\operatorname*{máx}_{i''} \log\left(p(i'')p(c|i'')p(\boldsymbol{x}|i'')\right) + \\ &\log \sum_i \exp\left(\log p(i)p(c|i)p(\boldsymbol{x}_n|i) - \operatorname*{máx}_{i''} \log\left(p(i'')p(c|i'')p(\boldsymbol{x}_n|i'')\right)\right) \end{aligned} \tag{6.7}$$

6.2. Experimentación

En esta sección se presentan los resultados obtenidos utilizando la regla de clasificación expuesta en la sección 6.1 tras una etapa de aprendizaje no supervisado. Se ha utilizado la misma técnica para determinar la condición de parada del entrenamiento que aplicamos en el capítulo 5, empleando un conjunto de validación para determinar el número de iteraciones del algoritmo mediante validación cruzada, para después utilizar todo el conjunto de entrenamiento para determinar el valor de los parámetros. En este caso, sin embargo, no fue necesario repetir los experimentos variando la semilla inicial, ya que la variabilidad producida por la aleatoriedad de la inicialización no era tan acentuada como en el caso supervisado. Este hecho se puede explicar de una forma intuitiva si nos percatamos de que a la hora de realizar la inicialización, la selección de los prototipos se lleva a cabo explorando todos los documentos de entrenamiento, en lugar del subconjunto correspondiente a una clase, como pasaba en el capítulo 5. Esto provoca que exista una mayor diversidad del tipo de documentos o lo que es equivalente, una mayor distancia entre ellos, con lo que la elección del prototipo inicial no influye de forma tan determinante.

6.2.1. EuTrans

En este caso, la elección del tipo de suavizado no tiene una gran influencia sobre el resultado obtenido, por lo tanto en la exposición nos centraremos en el suavizado backoff unigrama, por ser el que produce un mejor resultado, pero las conclusiones son extrapolables al resto de técnicas de suavizado.

En la figura 6.1 se muestra la evolución del error al aumentar el número de clases. Se observa que este parámetro tiene una influencia decisiva sobre la efectividad del clasificador. Podemos observar que utilizando únicamente 4 clases, es decir, el número original de clases presentes en el corpus, el error obtenido es del $48.40\,\%$. Éste es un resultado ciertamente pobre comparado con los obtenidos en capítulos anteriores, pero demuestra que el algoritmo EM ha sido capaz de detectar una cierta estructura dentro del conjunto de entrenamiento que se corresponde con la que habíamos definido con anterioridad[1]. Si aumentamos el número de clases, la tasa de error baja de forma drástica hasta llegar a un punto en el que se estabiliza, a partir del cual seguir aumentando el número de componentes no tiene un efecto significativo. En concreto, la menor tasa de error se obiene utilizando 68 componentes. Esta es una cantidad más elevada de lo que se podría esperar en un principio teniendo en cuenta los resultados presentados en 5.5.1, en los que el menor error se conseguía utilizando 6 o 7 componentes por clase, es decir, 24 o 28 componentes en total.

Al igual que sucede con el tipo de suavizado utilizado, la elección del parámetro de suavizado, no tiene una gran influencia sobre la tasa de error, como muestra la figura 6.2 (nótese la escala del eje y).

[1] Nótese que si siguieramos un procedimiento de clasificación meramente aleatorio la tasa de error esperada sería del $75\,\%$.

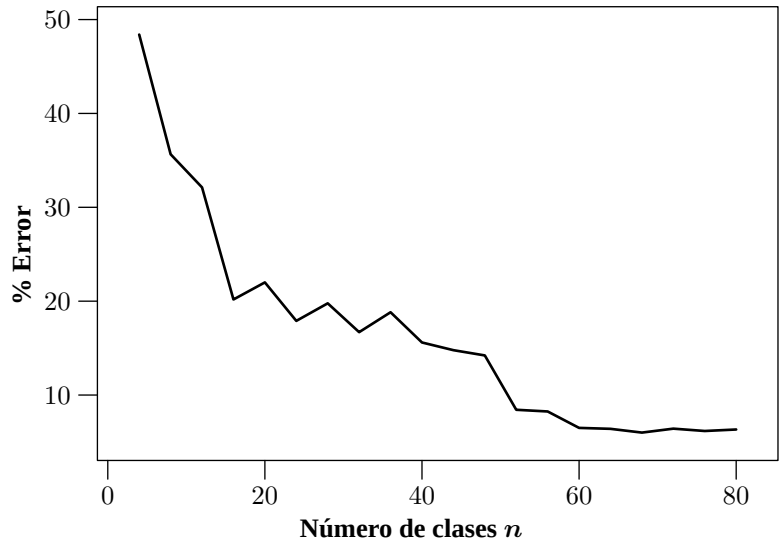

FIGURA 6.1: Evolución del error variando el número de clases para el corpus Eutrans, utilizando suavizado backoff unigrama, con $b = 0.8$

El mejor resultado para este corpus se obtiene utilizando suavizado backoff unigrama con $b = 0.8$ y 68 componentes no supervisadas. La tasa de error conseguida en este caso es del $6.00\,\%$.

6.2.2. WebKB

Para este corpus de datos, aunque el comportamiento es similar para todas las técnicas de suavizado, el método concreto que empleemos sí tiene una influencia importante sobre el valor de la tasa error, como se muestra en la figura 6.3. De nuevo podemos fijarnos en el valor obtenido para 4 componentes para comprobar que el algoritmo ha sido capaz de detectar un cierta estructura en el corpus. El mejor valor se obtiene sin embargo al utilizar un número mayor de componentes, en concreto 16. También se observa que si aumentamos el número de clases no supervisadas por encima de este valor, la tasa de error aumenta para luego alcanzar un intervalo en el que permanece relativamente estable.

El parámetro de suavizado elegido también influye en los resultados, como muestra la figura 6.4. Un resumen de los resultados obtenidos se muestra en la tabla 6.1. Se puede observar que son ciertamente inferiores a los obtenidos anteriormente (ver tabla 5.2, pág. 44). Esto es comprensible ya que nos hemos propuesto una tarea mucha más ambiciosa, el aprendizaje de forma no supervisada de los parámetros de una mixtura de multinomiales, es decir, ignoramos la información, claramente relevante, que nos pueda proporcionar un etiquetado previo.

FIGURA 6.2: Evolución del error sobre el corpus Eutrans variando el parámetro b de suavizado. El número de componentes es 68.

FIGURA 6.3: Evolución del error variando el número de clases para el corpus WebKB.

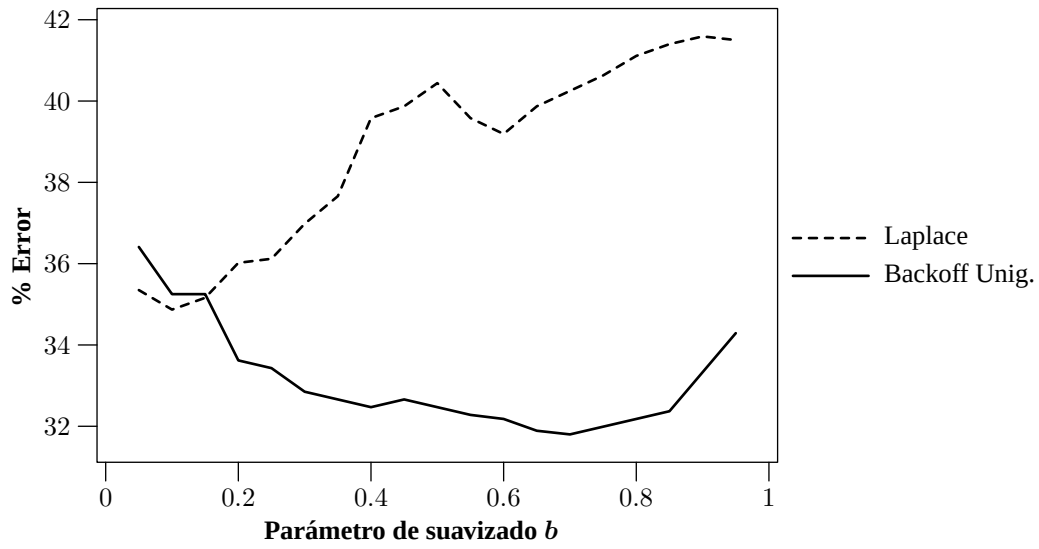

FIGURA 6.4: Evolución del error variando el parámetro de suavizado para el corpus WebKB.

Suavizado	**Nº de clases**	**Parámetro *b***	**% Error**
Laplace	16	0.1	34.87 %
Backoff Unif.	16	0.45	34.77 %
Backoff Unig.	16	0.7	31.80 %
Interpolación Unif.	16	0.55	34.20 %
Interpolación Unig.	16	0.95	32.76 %

TABLA 6.1: Mejores resultados obtenidos para el corpus WebKB.

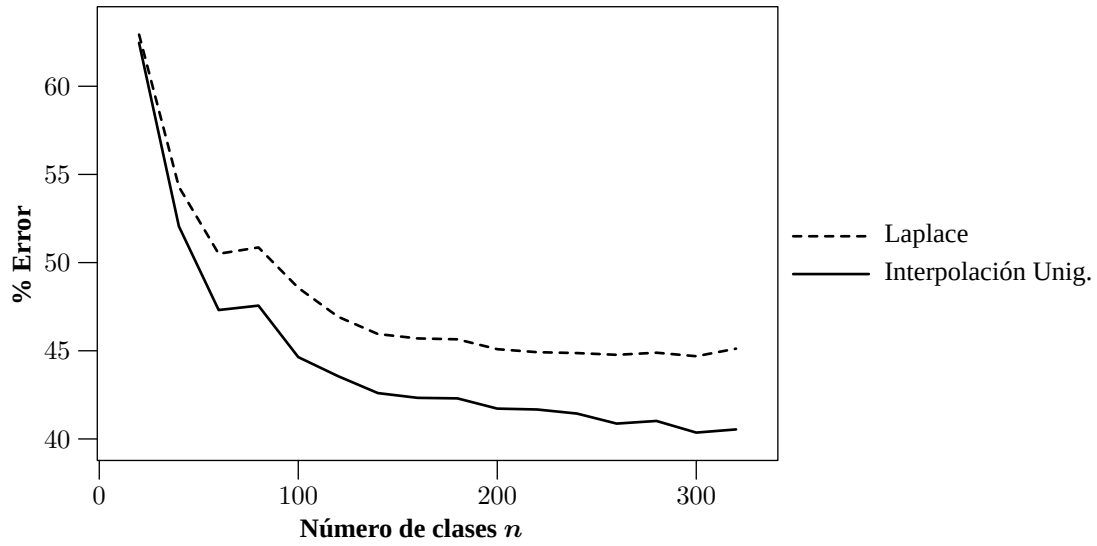

FIGURA 6.5: Evolución del error variando el número de clases para el corpus 20 newsgroups.

6.2.3. 20 newsgroups

Utilizando el corpus 20 newsgroups nuevamente comprobamos que los distintos tipos de suavizado tienen un comportamientos similar, aunque la utilización de uno u otro influye de forma determinante sobre la tasa de error obtenida (figura 6.5). En esta caso, al igual que sucedía con el corpus EuTrans, la tasa de error disminuye a medida que aumentamos el número de componentes no supervisadas de la mixtura hasta llegar a un valor en el que permanece prácticamente estable. El efecto del valor del parámetro de suavizado también es importante (figura 6.6).

Los mejores resultados obtenidos con cada método de suavizado se muestran en la tabla 6.2. El menor error conseguido es de aproximadamente un $40\,\%$, un valor quizás no del todo satisfactorio, pero sí aceptable teniendo en cuenta la dificultad de la tarea. Si analizamos la matriz de confusión presentada en la figura 6.7 comprobamos que el algoritmo ha sido capaz de distinguir la categoría generales, pero encuentra problemas al tratar de detectar la separación entre grupos similares. Un ejemplo claro lo proporcionan los distintos grupos de noticias dedicados a la informática (“esquina superior izquierda” de la matriz) o aquellos que tratan temas de política (“esquina inferior derecha”). También es notable destacar que el grupo `misc.forsale` es el que presenta una mayor dispersión entre el resto de grupos. Esto es comprensible, ya que se pueden vender multitud de objetos y en particular, es habitual la aparición de venta de productos informáticos de segunda mano, por lo que encontrar un prototipo adecuado para esta clase entraña una elevada dificultad.

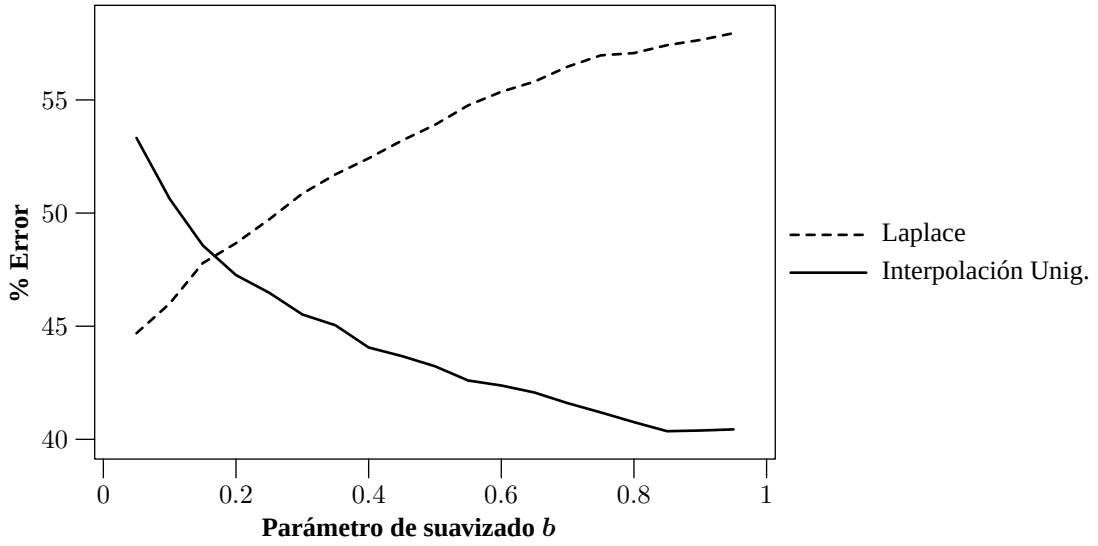

FIGURA 6.6: Evolución del error variando el parámetro de suavizado para el corpus 20 newsgroups.

Suavizado	**Nº de clases**	**Parámetro *b***	**% Error**
Laplace	300	0.05	44.69 %
Backoff Unif.	280	0.75	45.60 %
Backoff Unig.	300	0.7	40.76 %
Interpolación Unif.	300	0.75	45.80 %
Interpolación Unig.	300	0.85	40.36 %

TABLA 6.2: Mejores resultados obtenidos para el corpus 20 newsgroups.

	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	141	3	0	0	0	3	0	0	0	0	0	0	0	0	0	21	3	2	13	3
comp.graphics	0	138	3	12	4	16	3	1	1	0	0	5	4	10	2	1	0	0	0	0
comp.os.ms-windows.misc	0	62	10	35	25	27	3	0	3	1	0	3	0	7	0	0	0	0	6	1
comp.sys.ibm.pc.hardware	0	18	9	125	24	9	0	0	0	0	0	1	11	0	0	0	0	0	0	3
comp.sys.mac.hardware	0	29	4	75	52	7	1	0	5	0	0	1	10	8	0	0	2	0	5	0
comp.windows.x	0	31	2	5	4	150	0	0	0	0	0	2	0	2	2	0	0	0	0	0
misc.forsale	0	40	4	39	24	15	12	8	4	0	11	0	24	14	1	0	1	0	2	1
rec.autos	0	8	3	2	0	0	0	144	2	0	0	0	26	8	0	0	2	1	3	1
rec.motorcycles	0	5	0	0	1	4	0	20	123	0	1	1	28	4	0	0	5	0	6	2
rec.sport.baseball	0	1	0	0	0	1	2	2	0	142	43	0	0	5	0	0	1	0	3	0
rec.sport.hockey	0	0	0	0	0	0	1	0	0	0	193	0	0	1	1	0	0	0	4	0
sci.crypt	2	9	0	0	2	2	5	0	0	0	0	162	2	7	1	0	3	0	3	0
sci.electronics	0	38	3	13	11	8	3	0	1	0	0	5	94	16	5	1	1	0	0	1
sci.med	7	8	1	0	0	3	0	1	6	0	0	1	9	141	4	6	4	2	6	1
sci.space	0	9	0	0	0	0	2	1	0	0	0	2	7	14	156	0	0	1	8	0
soc.religion.christian	10	3	0	1	0	2	1	0	0	0	1	1	0	6	0	165	0	1	3	3
talk.politics.guns	1	0	0	0	0	1	0	0	0	0	0	8	1	9	1	2	155	5	12	4
talk.politics.mideast	5	1	0	0	0	1	1	0	0	0	1	0	0	17	1	9	3	130	30	1
talk.politics.misc	4	1	0	0	0	1	0	0	4	0	2	6	0	4	2	7	65	10	85	9
talk.religion.misc	59	1	0	0	0	3	0	0	2	0	0	3	0	7	7	37	17	3	9	52

FIGURA 6.7: Matriz de confusión para el corpus 20 newsgroups.

CAPÍTULO 7

Conclusiones

En este proyecto hemos abordado la tarea de clasificación de textos utilizando distintas técnicas de aprendizaje basadas en el paradigma de máxima verosimilitud. Hemos prestado especial interés en la aplicación de técnicas de suavizado y en la influencia que los distintos parámetros tienen sobre la efectividad de los clasificadores.

En el capítulo 3 estudiamos un clasificador multinomial, un modelo relativamente simple pero que consigue buenos resultados. Presentamos las distintas técnicas de suavizado que emplearíamos a lo largo del proyecto y comprobamos la eficacia de este clasificador sobre los distintos corpora de datos.

En el capítulo 4 hemos presentado el conocido algoritmo "Expectation Maximization" (EM) y lo hemos particularizdo al caso en el que el modelo a tratar es una mixtura de multinomiales. En el capítulo 5 hemos utilizado una mixtura como modelo para cada clase supervisada y empleado el algoritmo EM para estimar los parámetros. En dos de los tres corpus hemos obtenido una mejora utilizando una mixtura frente al modelo simple de una única multinomial por clase. Para el corpus 20 newsgroups hemos conseguido una leve mejora para algunos casos específicos, pero el mejor resultado obtenido en este capítulo no supera al que obtuvimos anteriormente.

En el capítulo 6 nos proponíamos un objetivo mucho más ambicioso, comprobar si mediante un aprendizaje completamente no supervisado el algoritmo EM lograba detectar la estructura que habíamos definido previamente en cada uno de los corpus. Hemos comprobado que para la tarea EuTrans sí se logra una tasa de error satisfactoria. Para WebKB y 20 newsgroups la efectividad conseguida era bastante menor, debido a que estas tareas presentan una complejidad más elevada. En concreto para el corpus 20 newsgroups la división entre algunos grupos de noticias es ciertamente difusa, por lo que no incluir la información que aporta el etiquetado inicial supone una penalización importante. De todas formas los resultados obtenidos demuestran que el algoritmo EM efectivamente logra encontrar una división adecuada a grandes rasgos, aunque

quizá no tan fina como la que desearíamos en un principio.

Como ya comentábamos en el capítulo 2 hemos omitido intencionadamente de la tarea una selección de características basada en métodos que presuponen un etiquetado de las muestras de entrenamiento, teniendo en mente sobre todo el aprendizaje no supervisado del capítulo 6. Reducir la cantidad de características de las muestras y, por lo tanto, el número de parámetros a estimar en los modelos, puede tener un impacto significativo en un clasificador, como muestran por ejemplo [YP97] o [LJ02], por lo que una línea de trabajo futura podría consistir en integrar los métodos de selección de características en las técnicas de aprendizaje que hemos estudiado en este proyecto fin de carrera.

Apéndices

“¿Sabes sumar?”, le preguntó la Reina Blanca.
“¿Cuántas son una y una y una y una y una y una y una y una y una y una?”
“No lo sé”, respondió Alicia. “He perdido la cuenta.”
“Así que no sabe sumar”, le interrumpió la Reina Roja.

“A través del espejo”, LEWIS CARROLL

APÉNDICE A

Archivos de biblioteca

En este capítulo detallaremos las funciones y tipos de datos que se han desarrollado, pero sin entrar en los detalles de implementación. Las funciones que en realidad están implementadas mediante macros se marcan mediante Macro. Hay que tener cuidado con los efectos secundarios al utilizar estas funciones si en la llamada se modifica algún argumento, por ejemplo mediante el operador ++.

A.1. Archivo `util.h`

En este archivo se encuentran las declaraciones de tipos de datos y funciones de utilidad general, no específicos para esta tarea. La implementación se encuentra en el arhivo `util.c`

- esespacio(**int** *c*): Devuelve cierto si *c* es un espacio. Macro
- Tipo de datos **t_buffer**:

  Es un tipo de datos diseñado como buffer de cadenas, por tanto encapsula internamente un array de tipo **char**∗. Se accede a la posición actual por medio del miembro *pos_buf*. La gestión de memoria dinámica se realiza automáticamente y se garantiza que una vez que se ha utilizado una dirección de memoria esta no cambiará a lo largo de la ejecución del programa, por lo que los punteros que se fijen continuarán siendo válidos. No permite operaciones para liberar memoria. Las funciones que ofrece son:

  - **char** ∗crea_buffer(**t_buffer** ∗*b*, **int** *tam*, **int** *max_len*): Inicializa el buffer *b* con un tamaño inicial de *tam* caracteres y preparado para almacenar cadenas de longitud máxima *max_len*.

  - **void** inc_buffer(**t_buffer** ∗*b*, **int** *len*): Incrementea la posición actual del buffer *b* en *len* caracteres.

- **int** lee_pal_fd(**char** ∗*buf*, **FILE** ∗*fd*): lee una palabra[1] del archivo *fd*, la almacena en *buf* (incluyendo un `'\0'` final) y devuelve el separador encontrado (*EOF* es un valor posible).
- **int** lee_pal(**char** ∗*buf*): Similara a la anterior, pero leyendo de la entrada estándar. Macro
- **void** mt(**void** ∗*p*): "Memory test", aborta la ejecución del programa si *p* es *NULL*, indicando el fichero y la línea desde donde se invocó. Macro
- **double** rand_in(**double** *min*, **double** *max*): Devuelve un número aleatorio entre *min* y *max*. Macro
- **int** rand_ent(**int** *min*, **int** *max*): Análoga a la anterior, pero trabajando con valores enteros. Macro
- **t_tipo** ∗∗matriz2D(**int** *x*, **int** *y*, **t_tipo**): reserva espacio para una matriz bidimensional de dimensiones $x \times y$ de tipo **t_tipo**. Macro
- **void** libera2D(**void** ∗∗*m*): libera el espacio reservado por la función anterior. Macro

A.2. Archivo `hash.h`

Este archivo contiene las declaraciones necesarias para la implementación de un diccionario llevada a cabo en el fichero `hash.c`. Internamente se trata de una tabla hash para pares (**char** ∗*palabra*, **int** *posicion*). Contiene un tipo de datos **t_hash** y las cabeceras de las funciones para tratar con él.

- **t_hash** ∗crea_hash(**t_hash** ∗*h*, **int** *tam*): inicializa una tabla hash de *tam* posiciones.
- **int** inserta_hash(**t_hash** ∗*hash*, **char** ∗*pal*): inserta la palabra *pal* en la tabla hash y devuelve la posición asignada.
- **int** inserta_hash_pos(**t_hash** ∗*hash*, **char** ∗*pal*, **int** *pos*): inserta la palabra *pal* con la posición asociada *pos*. Si la palabra ya existiera con otra posición se sobreescribe[2].
- **int** busca_hash(**t_hash** ∗*hash*, **char** ∗*pal*): devuelve la posición asociada a la palabra *pal*, o -1 si no existe.

[1] Es decir, una cadena de símbolos separada por algún tipo de blanco (espacio, salto de línea, tabulador...)

[2] Un uso de esta función que no respete la secuencialidad de las entradas puede provocar un mal funcionamiento de la tabla.

- **int** get_maxpos_hash(**t_hash** *h*): define el la última posición asignada, que coincide con el número de elementos almacenados en la tabla[3]. Macro
- **void** escribe_hash(**t_hash** ∗*h*, **FILE** ∗*fd*): escribe los elementos de la tabla hash *h* en el fichero *fd*, en formato ASCII. Macro
- **void** escribe_hash_pos(**t_hash** *h*, **FILE** *fd*): similar a la anterior, pero escribiendo además la posición asociada a cada elemento. Macro
- **void** escribe_hash_bin(**t_hash** ∗*h*, **FILE** ∗*fd*)
- **void** escribe_hash_pos_bin(**t_hash** ∗*h*, **FILE** ∗*fd*): similares a las dos anteriores, pero utilizando un formato binario. Macro

A.3. Archivo vector_disperso.h

Este es una archivo peculiar. Se ha llevado a cabo una implementación de un vector disperso, independiente del tipo de datos que encapsula, es decir, se ha intentado implementar un sencillo polimorfismo en lenguaje C. Todo el archivo se compone de una única macro que al ser invocada crea el tipo de datos y las funciones necesarias para manipularlo. Por ello, antes de poder utilizar el vector disperso se debe realizar una llamada a vector_disperso(**t_tipo**) al principio del archivo en el que se desea emplear, indicando un tipo de datos, por ejemplo

```
vector_disperso(int);
```

Esta llamada crearía el tipo de datos **vdisp_int** y las funciones necesarias para manípularlos.

Para entender correctamente la utilización de este tipo de datos necesitamos distinguir los conceptos de posición e índice en un vector disperso. En la figura A.1 se muestra un vector disperso de seis elementos. Por *posición* denotaremos la posición lógica de una elemento dentro de un vector, así, para el ejemplo tenemos las siguientes posiciones con sus valores asociados: $v[1]$=1, $v[5]$=12, $v[95]$=8, $v[150]$=63, $v[423]$=2 y $v[1000]$=11. El *índice* sin embargo es la posición dentro de la implementación del vector disperso. Éste índice se gestiona automáticamente, pero algunas funciones lo necesitan como parámetro para especificar a qué elemento se está accediendo.

Las funciones[4] que crea esta macro son

- **void** crea_vd(**vdisp** ∗*v*, **int** *tam*): crea un vector con espacio (inicial) para *tam* elementos.
- **void** anyade_vd(**vdisp** ∗*v*, **int** *pos*, **t_tipo** *val*): añade a *v* el elemento *v*[*pos*] = *val*.
- **int** pos_vd(**vdisp** ∗*v*, **int** *i*): devuelve la posición del elemento de índice *i*.

[3] El primer elemento recibe la posición 1.

[4] El nombre real de las funciones tiene el sufijo "*_tipo*" añadido al nombre que aquí se presenta.

posición	1	5	95	150	423	1000
valor	1	12	8	63	2	11
índice	1	2	3	4	5	6

FIGURA A.1: Ejemplo de un vector disperso.

- **t_tipo** val_vd(**vdisp** $*v$, **int** i): devuelve el valor del elemento de índice i.
- **void** pon_val_vd(**vdisp** $*v$, **int** i, **t_tipo** val): fija el valor del elemento de índice i en val.
- **int** len_vd(**vdisp** $*v$): devuelve la longitud de v.
- **void** vacia_vd(**vdisp** $*v$): vacía el vector v (sin liberar memoria).
- **void** libera_vd(**vdisp** $*v$): libera la memoria de v.

A.4. Archivo `comun.h`

En este archivo se reunen las funciones, tipos de datos y constantes que serán comunes para los programas `multinomial` (apéndice B) y `em` (apéndice C).

A.4.1. Constantes

Se declaran distintas constantes, como el tamaño de la tabla hash, el tamaño del buffer o el valor que se considerará $-\infty$.

A.4.2. Tipos de datos

Se declara el tipo de datos “documento” como

```
typedef struct
{
  char *nombre;
  int clase;
  vdisp_float v; /* float para poder tener normalización de longitud */
  int len;
} t_doc;
```

y el tipo “array de documentos” como

```
typedef struct
{
  t_doc *doc;
  int n_doc;
  int inc; /* Variable para la gestión de memoria */
  int n_reservados; /* ídem */
} t_arr_doc;
```

A.4.3. Funciones

Se definen las funciones

- **int** lee_dicc(**t_hash** **dicc*, **t_buffer** **buf*, **char** **fname*): lee un diccionario de *fname* guardando el resultado en *dicc*.
- **int** lee_linea_siw(*doc*, *dicc*, *anyade_dicc*, *h_clases*, *buf*): lee una lína en formato siw de la entrada estandar. Macro
- **int** lee_linea_siw_fd(**t_doc** **doc*, **t_hash** **dicc*, **int** *anyade_dicc*, **t_hash** **h_clases*, **t_buffer** **buf*, **FILE** **fd*): lee una línea en formato siw del archivo *fd*.
- **void** crea_arr_doc(**t_arr_doc** **arr*, **int** *tam*): crea un array de documentos con espacio (inicial) para *tam* documentos.
- **t_doc** **nuevo_doc(**t_arr_doc** **arr*): devuelve un nuevo documento vacío del array *arr*.

APÉNDICE B

Programa `multinomial.c`

En este apéndice se muestra la utilización del programa `multinomial`. La implementación se omite, por seguir un diseño similar al del programa `em` mostrado en el apéndice C y tratarse de una implementación más simple[1].

B.1. Utilización

El programa se invoca mediante el comando `multinomial` y acepta las siguientes opciones:

`-tr,--training <nom_fich>` Especifica el corpus de entrenamiento. Si se omite este argumento se leerá de la entrada estándar.

`-din,--diccin <nom_fich>` Especifica un diccionario de entrada.

`-dout, --diccout <nom_fich>` Si no se ha especificado un diccionario de entrada al utilizar esta opción se guardará el diccionario generado a partir de las muestras en el fichero que se especiique.

`-te,--test <nom_fich>` Especifica el corpus de test.

`-n,--names <nom_fich>` Muestra los nombres de las clases en lugar de los índices.

`-addeps, --laplace <epsilon>` Suavizado de Laplace.

`-adb0,--unifbackoff <b>` Suavizado uniform backoff.

[1] El programa completo se puede encontrar en el CD adjunto a la memoria (ver apéndice E).

`-adb1,--unigbackoff <b>` Suavizado unigram backoff.

`-adi0,--unifinter <b>` Suavizado uniform interpolation.

`-adi1,--uniginter <b>` Suavizado unigram interpolation.

`-a,--ascii <nom_fich>` Almacena el clasificador en formato ASCII en el fichero especificado.

`-b,--bin <nom_fich>` Almacena el clasificador en formato binario.

`-h,--help` Muestra una pequeña ayuda.

APÉNDICE C

Programa em.c

En este apéndice presentamos el programa principal que se ha desarrollado para este proyecto, la implementación del algoritmo EM, tanto para el caso supervisado como para el caso no supervisado.

C.1. Utilización

El programa se invoca mediante el comando `em` y acepta las siguientes opciones:

`-n, -nclases <num>` Especifica el número de clases no supervisadas. *Es obligatorio asignar una valor a este argumento*.

`-c, -clases-sup <num>` Indica que se utilizará aprendizaje supervisado con `<num>` clases supervisadas

`-tr, -training <nom_fich>` Especifica el fichero del que se leerá el conjunto de training en el caso de aprendizaje no supervisado o el "nombre base" si se emplea aprendizaje superviado. En este último caso los ficheros que se leerán serán `<nom_fich>1`, `<nom_fich>2`, ..., `<nom_fich>C`.

`-d, -dicc <nom_fich>` Especifica un diccionario de entrada.

`-te, -test <nom_fich>` Especifica un conjunto de test.

`-val, -validacion <nom_fich>` Especifica un conjunto de validación.

`-fval, -fake-val <nom_fich>` Especifica un "conjunto de validación" sobre el que se comprobará el error en cada iteración, pero que no detendrá la ejecución del algoritmo.

`-i, -iter <num>` Especifica el máximo número de iteraciones que se ejecutarán.

`-fi, -forceiter <num>` Fuerza la ejecución de un número determinado de iteraciones.

`-norm` Utiliza normalización de la longitud de los documentos.

`-ri, -randini` Inicialización aleatoria.

`-mi, -muestraini` Inicialización basada en la selección aleatoria de muestras.

`-maxminini` Inicialización mediante el algoritmo maxmin (opción por defecto).

`-s, -seed <num>` Especifica una semilla (si no se especifica o se le asigna un valor 0 se utiliza utiliza `time(NULL)`).

`-addeps, -laplace <epsilon>` Suavizado de Laplace.

`-adb0,--unifbackoff <b>` Suavizado uniform backoff.

`-adb1,--unigbackoff <b>` Suavizado unigram backoff.

`-adi0,--unifinter <b>` Suavizado uniform interpolation.

`-adi1,--uniginter <b>` Suavizado unigram interpolation.

`-i<suav> <b>` Especificando una i inicial (por ejemplo -iadb0) se aplica suavizado "intermedio", es decir, suavizado en cada una de las iteraciones del EM.

`-h, -help` Muestra una pequeña ayuda.

C.2. Implementación

En esta sección se incluye el código desarrolado en lenguaje C.

> En la implementación denotaremos por $i = 1, \ldots, I$ las clases supervisadas y por $c = 1, \ldots, C$ las no supervisadas, en contra de la práctica habitual a lo largo del presente proyecto, debido a que ésta fue la notación escogida originalemente para la implementación.

C.2.1. #include's y constantes

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <values.h>
#include <math.h>
#include <string.h>

/* Las librerías que hemos definido anteriormente */
#include "util.h"
#include "comun.h"
#include "hash.h"

#define -∞_d  -DBL_MAX /* -∞ en double */
#define +∞_i   INT_MAX /* +∞ en int */

/* Máximo número de iteraciones por defecto */
#define MAX_ITER 50

/* Flags para controlar el funcionamiento */
typedef unsigned int t_flags;
#define FL_DICC            (1<<0)
#define FL_ESCRIBE         (1<<1)
#define FL_TEST            (1<<2)
#define FL_LAPLACE         (1<<3)
#define FL_BACKOFF         (1<<4)
#define FL_INTERPOLATION   (1<<5)
#define FL_UNIFORM         (1<<6)
#define FL_UNIGRAM         (1<<7)
#define FL_INTER_SMOOTH    (1<<8)
#define FL_END_SMOOTH      (1<<9)
#define FL_SMOOTH          (FL_INTER_SMOOTH | FL_END_SMOOTH)
#define FL_NORM            (1<<10)
#define FL_RAND_INI        (1<<11)
#define FL_MUESTRA_INI     (1<<12)
#define FL_MAXMIN_INI      (1<<13)
#define FL_VAL             (1<<14)
#define FL_FAKE_VAL        (1<<15)
#define FL_ESCRIBE_DICC    (1<<16)
#define FL_NO_SUP          (1<<17)
#define FL_FORCE_ITER      (1<<18)

#define EPSILON_INI 0.05 /* ε para suavizar la inicialización */
```

C.2.2. Macros y funciones auxiliares

```
/* Cálculo de la clase supervisada asociada a una no supervisada. */
#define superv(c) ((int) ceil((float) (c)/n_clases))
/* y viceversa */
#define no_superv(c, i) (((i)−1)∗n_clases + c)
/* Cálculo de la clase no supervisada dentre de una clase supervisada */
#define no_superv_sup(c, i) ((c) − ((i)−1)∗n_clases)
/* Estas son funciones muy específicas (dependen del nombre de la variable n_clases), por lo
que deberían usarse con cuidado */

/* Función auxiliar para el intercambio de dos matrices bidimensionales */
void intercambia_dmat2D(double ∗∗∗m1, double ∗∗∗m2)
{
  double ∗∗temp;
  temp = ∗m1;
  ∗m1 = ∗m2;
  ∗m2 = temp;
}
```

C.2.3. Cálculo del error de clasificación

Caso supervisado

Para el caso supervisado utilizamos la ecuación desarrollada en 5.2

$$g_i(\boldsymbol{x}_n) = \log p(i) + \max_{c''}\left(\log p(c'') + \sum_d x_{nd}\log p_{cd}\right) + \log\sum_c \exp\left[\log p(c) + \sum_d x_{nd}\log p_{cd} - \max_{c''}\left(\log p(c'') + \sum_d x_{nd}\log p_{c''d}\right)\right]$$

```
double calcula_error_sup(t_arr_doc ∗arr_te, int n_clases, int n_clases_sup,
                         double ∗p_i, double ∗p_c, double ∗∗p_cd,
                         t_flags flags, int muestra)
{
  double ∗logpisum, ∗logpcsum;
  double suma, max, maxclas;
  int i, c, c_sup, x, d, ind_maxclas, errores;
  t_doc ∗doc;

  mt(logpisum = (double ∗) malloc((n_clases_sup+1) ∗ sizeof(double)));
```

```
  mt(logpcsum = (double *) malloc((n_clases+1) * sizeof(double)));
  errores = 0;

  for (x=0; x<arr_te→n_doc; x++) {
    doc = &arr_te→doc[x];
    for (i=1; i≤n_clases_sup; i++) {
      for (c=no_superv(1, i); c≤no_superv(n_clases, i); c++) {
        c_sup = no_superv_sup(c, i);
        logpcsum[c_sup] = log(p_c[c]);
        for (d=0; d<doc→len; d++)
            logpcsum[c_sup] += val_vd_float(&doc→v, d) *
              log(p_cd[c][pos_vd_float(&doc→v, d)]);
      }
      max = -∞_d;
      for (c=no_superv(1, i); c≤no_superv(n_clases, i); c++) {
        c_sup = no_superv_sup(c, i);
        if (logpcsum[c_sup] > max)
          max = logpcsum[c_sup];
      }
      suma = 0;
      for (c=no_superv(1, i); c≤no_superv(n_clases, i); c++)
        suma += exp(logpcsum[no_superv_sup(c, i)] - max);
      logpisum[i] = log(p_i[i]) + max + suma;
    }

    maxclas = -∞_d;
    ind_maxclas = 1;
    for (i=1; i≤n_clases_sup; i++)
      if (logpisum[i] ≥ maxclas) {
        maxclas = logpisum[i];
        ind_maxclas = i;
      }

    if (doc→clase ≠ ind_maxclas)
      errores++;
    if (muestra)
      printf("%d\t%d\n", doc→clase, ind_maxclas);
  }

  free(logpisum);
  free(logpcsum);
  return (double) errores/arr_te→n_doc;
}
```

Caso no supervisado

Para el caso no supervisado utilizamos la ecuación desarrollada en 6.1

$$g_i(\boldsymbol{x}) = \max_{c''} \log\left(p(c'')p(i|c'')p(\boldsymbol{x}|c'')\right) + \log \sum_{c} \exp\left(\log(p(c)p(i|c)p(\boldsymbol{x}|c)) - \max_{c''} \log(p(c'')p(i|c'')p(\boldsymbol{x}|c''))\right)$$

```
double calcula_error_nosup(t_arr_doc *arr, t_arr_doc *arr_te,
                           double *p_c, double **p_cx, double **p_cd,
                           int n_clases_sup, int n_clases,
                           unsigned int flags, int muestra)
{
  double **p_ic, *Nc, *clasc;
  double suma, max, maxclas;
  int c, x, d, i, ind_maxclas, errores;
  t_doc *doc;

  mt(p_ic=cmatriz2D(n_clases_sup+1, n_clases+1, double));
  mt(Nc=(double *) calloc((n_clases+1),sizeof(double)));
  mt(clasc=(double *) malloc((n_clases+1)*sizeof(double)));
  /* Calculamos p(i|c) = 1/(sum_n p(c|x_n)) * sum_{n:i_n=i} p(c|x_n) */
  for (c=1; c<=n_clases; c++) {
    for (x=0; x<arr->n_doc; x++) {
      Nc[c] += p_cx[c][x]; /* Denominador */
      p_ic[arr->doc[x].clase][c] += p_cx[c][x];
    }
  }
  for (i=1; i<=n_clases_sup; i++)
    for (c=1; c<=n_clases; c++)
      p_ic[i][c] /= Nc[c];

  errores = 0;
  for (x=0; x<arr_te->n_doc; x++) {
    doc = &arr_te->doc[x];
    ind_maxclas = 1;
    maxclas = -inf;
    for (i=1; i<=n_clases_sup; i++) {
      max = -inf;
      for (c=1; c<=n_clases; c++) {
```

```
            suma = log(p_c[c]*p_ic[i][c]);
            for (d=0; d<doc→len; d++)
              suma += val_vd_float(&doc→v, d) * log(p_cd[c][pos_vd_float(&doc→v, d)]);
            if (suma > max)
              max = suma;
            clasc[c] = suma;
          }
          suma = 0;
          for (c=1; c≤n_clases; c++)
            suma += exp(clasc[c] − max);
          suma = log(suma) + max;
          if (suma > maxclas) {
            ind_maxclas = i;
            maxclas = suma;
          }
        }
        if (doc→clase ≠ ind_maxclas)
          errores++;
        if (muestra) {
          printf("%d\t%d\n", doc→clase, ind_maxclas);
        }
      }

      libera2D(p_ic);
      free(Nc);
      free(clasc);

      return (double) errores/arr_te→n_doc;
}
```

C.2.4. Funciones de E/S

Lectura de un array de documentos

```
void lee_arr_doc(t_arr_doc *arr, t_hash *dicc, int anyade_dicc,
                 t_hash *clases, t_buffer *buf, char *fname)
{
  FILE *fd;

  if (¬(fd=fopen(fname, "r"))) {
    fprintf(stderr, "Error abriendo %s", fname);
    perror("");
```

```
    exit(1);
  }
  while (lee_linea_siw_fd(nuevo_doc(arr), dicc, anyade_dicc,
                          clases, buf, fd) ≥ 0);
  arr→n_doc−−;
  fclose(fd);
}
```

Escritura del clasificador

```
void escribe_clasif(int n_clases, int n_pal, double *p_c, double **p_cd,
                    char *fname)
{
  int c, d;

  printf("#Clases\t%d\n", n_clases);
  for (c=1; c≤n_clases; c++)
    printf("%d\tClase%d\n", c, c);
  printf("#Prob_Clases\n");
  for (c=1; c≤n_clases; c++)
    printf("%d\t%g\n", c, p_c[c]);
  printf("#Params\n");
  for (c=1; c≤n_clases; c++) {
    printf("#Clase %d\n", c);
    for (d=1; d≤n_pal; d++)
      /* if (p_cd[c][d] > 0) */ /* Disminuye el tamaño del fichero, pero hace que sea un poco
más difícil trabajar con él. */
      printf("%d %g\n", d, p_cd[c][d]);
  }
}
```

C.2.5. El algoritmo EM

Paso E

Cálculo del paso E del algoritmo EM, devuelve una aproximación de la log-verosimilitud. Utilizamos el cálculo robusto de probabilidades desarrollado en las sección 4.5.

```
double pasoE(t_arr_doc *arr, int n_clases, int n_clases_sup, int i,
             double *p_c, double **p_cd, double **p_cx, double *logpcsum)
{
  double logver, max_logpcsum, suma;
```

```
  int x, c, d;
  t_doc *doc;

  logver = 0.0;
  for (x=0; x<arr→n_doc; x++) {
    doc = &arr→doc[x]; /* Comodidad de escritura */
    max_logpcsum = -∞d;
    for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++) {
      logpcsum[c] = log(p_c[c]);
      for (d=0; d<doc→len; d++)
        logpcsum[c] += val_vd_float(&doc→v, d) * log(p_cd[c][pos_vd_float(&doc→v, d)]);
      if (logpcsum[c] > max_logpcsum)
        max_logpcsum = logpcsum[c];
    }
    suma = 0.0;
    for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++) {
      p_cx[c][x] = exp(logpcsum[c] - max_logpcsum);
      suma += p_cx[c][x];
    }
    for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++)
      p_cx[c][x] /= suma;
    logver += max_logpcsum + log(suma);
  }

  return logver;
}
```

Suavizado

Implementación de los distintos suavizados. p_{cd} contiene las *cuentas*. En $\tilde{p}_{cd}$ (qe puede ser igual a p_{cd}) se dejarán los valores suavizados

```
void suavizado(double **p_cd, double **p̃_cd, double *β, double par_smooth,
               int c, int n_clases, int n_pal, t_flags flags)
{
  int d;
  double M, suma, frac_smooth;

  if (flags & FL_LAPLACE) {
    /* p̃_cd = (ε + Σ_n p(c|x_n) x_nd) / (Σ_d (ε + Σ_n p(c|x_n) x_nd)) */
    for (d=1; d≤n_pal; d++)
```

```
      p̃cd[c][d] = (pcd[c][d] + par_smooth) / (pcd[c][0] + n_pal∗par_smooth);
    } else {
      /* La técnica de suzavizado es backoff o interpolation. Para ambos métodos la masa de
probabilidad ganada es M = (Σd' mín(b, Ncd')) / (Σd' Ncd') */
      M=0;
      suma = 0;
      frac_smooth = 0;
      for (d=1; d≤n_pal; d++) {
        if (pcd[c][d] ≤ par_smooth) {
          M += pcd[c][d];
          frac_smooth += β[d]; /* Si hacemos backoff */
        } else
          M += par_smooth;
        suma += pcd[c][d];
      }
      M /= suma;



      if (flags & FL_BACKOFF) {
                                  ⎧ (Ncd − b) / (Σd' Ncd')        si Ncd > b
        /*p̃cd = M ∗ β[d] / (Σd':Ncd'≤b βd')   si Ncd ≤ b   */
                                  ⎩
        for (d=1; d≤n_pal; d++)
          if (pcd[c][d] > par_smooth)
            p̃cd[c][d] = (pcd[c][d] − par_smooth) / pcd[c][0];
          else
            p̃cd[c][d] = M∗β[d]/frac_smooth;
      } else { /* if (flags & FL_INTERPOLATION) */
        for (d=1; d≤n_pal; d++) {
          if (pcd[c][d] > par_smooth)
            p̃cd[c][d] = (pcd[c][d] − par_smooth) / pcd[c][0] + M∗β[d];
          else
            p̃cd[c][d] = M∗β[d];
        }
      }
    }
  }
```

Inicialización

Cálculo de la distancia entre dos documento como la distancia de Hamming, considerando las cuentas > 0 como 1.

```
int distancia(t_doc *doc1, t_doc *doc2)
{
  int d1, d2, dist;

  dist = 0;
  for (d1 = 0; d1<doc1→len; d1++) {
    for (d2 = 0;
         d2<doc2→len ∧ pos_vd_float(&doc1→v, d1) ≠ pos_vd_float(&doc2→v, d2);
         d2++);
    if (d2 == doc2→len)
      dist++;
  }
  for (d2 = 0; d2<doc2→len; d2++) {
    for (d1 = 0;
         d1<doc1→len ∧ pos_vd_float(&doc2→v, d2) ≠ pos_vd_float(&doc1→v, d1);
         d1++);
    if (d1 == doc1→len)
      dist++;
  }

  return dist;
}

void inicializacion(t_arr_doc *arr_arr, double *p_i, double *p_c, double **p_cd,
                    int n_clases_sup, int n_clases, int n_pal, int max_n_doc,
                    t_flags flags)
{
  double suma;
  int i, c, x, d;
  int *Dp, dpq, maxmin, p;
  t_doc *doc;
  t_arr_doc *arr;

  if (flags & FL_MAXMIN_INI)
    mt(Dp = (int *) malloc(max_n_doc*sizeof(int)));
```

```
/* Probabilidad a priori de las clases supervisadas */
suma = 0;
for (i=1; i≤n_clases_sup; i++)
  suma += arr_arr[i].n_doc;
for (i=1; i≤n_clases_sup; i++)
  p_i[i] = (double) arr_arr[i].n_doc / suma;

for (i=1; i≤n_clases_sup; i++) {
  arr = &arr_arr[i];

  if (flags & FL_MAXMIN_INI) {
    for (x=0; x<arr→n_doc; x++)
      Dp[x] = +∞_i;
    x = −1;
  }

  for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++) {
    p_c[c] = (double) 1/n_clases;

    /* Inicialización aleatoria. Hay que tener en cuenta que ∑_d p_cd = 1 */
    if (flags & FL_RAND_INI) {
      suma = 0.0;
      for (d=1; d≤n_pal; d++) {
        p_cd[c][d] = rand_in(0,1);
        suma += p_cd[c][d];
      }
      for (d=1; d≤n_pal; d++)
        p_cd[c][d] /= suma;

    } else {
      /* Suavizado de los prototipos */
      for (d=1; d≤n_pal; d++)
        p_cd[c][d] = EPSILON_INI;

      if (flags & FL_MUESTRA_INI) {
        x = rand_ent(0, arr→n_doc−1);

      } else if (flags & FL_MAXMIN_INI) {
        if (x == −1) /* Primera iteración */
          x = rand_ent(0, arr→n_doc−1);
        else {
          maxmin = 0;
```

```
            for (p = 0; p < arr→n_doc; p++) {
              if (Dp[p] == −1)
                continue;
              dpq = distancia(&arr→doc[x], &arr→doc[p]);
              if (dpq < Dp[p])
                Dp[p] = dpq;
              if (Dp[p] > maxmin) {
                x = p;
                maxmin = Dp[p];
              }
            }
            Dp[x] = −1;
          }
        }

        doc = &arr→doc[x];
        for (d=0; d<doc→len; d++)
          p_cd[c][pos_vd_float(&doc→v, d)] += val_vd_float(&doc→v, d);
        for (d=1; d≤n_pal; d++)
          p_cd[c][d] /= doc→len + EPSILON_INI∗n_pal;
      }
    }
  }

  if (flags & FL_MAXMIN_INI)
    free(Dp);
}
```

El algoritmo

```
void em(t_arr_doc ∗arr_arr, int n_clases, int n_clases_sup, t_hash ∗clases_sup,
        int n_pal, int max_iter, t_flags flags, t_arr_doc ∗arr_te,
        t_arr_doc ∗arr_val, double par_smooth)
{
  double ∗p_i, ∗p_c, ∗∗p_cd, ∗∗p_cx, ∗logpcsum, ∗∗p̃_cd, ∗β, ∗ant_logver, ∗∗ant_pcd;
  double suma, logver, logver_total, ant_logver_total,
    error_tr, error_val, ant_error_val;
  int x, c, d, i, it, it_igual, max_n_doc, suavizados, n_clases_sup_real, salir;
  t_doc ∗doc;
  t_arr_doc ∗arr;

  max_n_doc = −1;
```

```
  for (i=1; i≤n_clases_sup; i++)
    if (arr_arr[i].n_doc > max_n_doc)
      max_n_doc = arr_arr[i].n_doc;

  if (flags & FL_NO_SUP)
    n_clases_sup_real = get_maxpos_hash(clases_sup);
  else
    n_clases_sup_real = n_clases_sup;

  /* Reservamos memoria */
  mt(p_c = (double *) malloc((n_clases*n_clases_sup + 1)*sizeof(double)));
  mt(p_cd = matriz2D(n_clases*n_clases_sup + 1, n_pal+1, double));
  mt(p_cx = matriz2D(n_clases*n_clases_sup + 1, max_n_doc, double));
  mt(logpcsum = (double *) malloc((n_clases*n_clases_sup + 1)*sizeof(double)));
  /* logpcsum no se utiliza en esta función, pero la declaramos aquí para evitar sucesivas
llamadas a malloc() en cada paso E. */
  mt(ant_logver = malloc((n_clases_sup + 1)*sizeof(double)));
  mt(p_i = malloc((n_clases_sup+1)*sizeof(double)));

  if (flags & FL_SMOOTH)
    mt(p̃_cd = matriz2D(n_clases*n_clases_sup + 1, n_pal+1, double));
  /* p̃_cd lo utilizaremos para almacenar valores provisionales. Si hay suavizado intermedio
almacenará los valores suavizados, si hay suavizado final (con validación) almacenará las
cuentas asociadas a las probabilidades. */
  if (flags & FL_VAL)
    mt(ant_pcd = matriz2D(n_clases*n_clases_sup + 1, n_pal+1, double));
  /* En ant_pcd almacenaremos los valores anteriores que corredpondan (cuentas o
probabilidades). */

  /* Inicialización del suavizado */
  if (flags & FL_UNIFORM ∨ flags & FL_UNIGRAM) {
    mt(β = malloc((n_pal+1)*sizeof(double)));
    /* El vector β lo utilizaremos para simplificar el tratamiento de adb[01] y adi[01] */
    if (flags & FL_UNIFORM) {
      for (d=1; d≤n_pal; d++)
        β[d] = (float) 1/n_pal;
    } else if (flags & FL_UNIGRAM){
      /* β_d = (Σ_n x_nd) / (Σ_d' Σ_n x_nd') */
      suma = 0.0;
      for (d=1; d≤n_pal; d++)
        β[d] = 0;
```

```
      for (i=1; i≤n_clases_sup; i++) {
        arr = &arr_arr[i];
        for (x=0; x≤arr→n_doc; x++) {
          doc = &arr→doc[x];
          for (d=0; d<doc→len; d++) {
            β[pos_vd_float(&doc→v, d)] += val_vd_float(&doc→v, d);
            suma += val_vd_float(&doc→v, d);
          }
        }
      }
      for (d=1; d≤n_pal; d++)
        β[d] /= suma;
    }
}

inicializacion(arr_arr, p_i, p_c, p_cd, n_clases_sup, n_clases, n_pal, max_n_doc, flags);

it = 1;
it_igual = 0;

/* Primero realizamos un paso E, tanto si suavizamos como si no */
logver_total = 0.0;
for (i=1; i≤n_clases_sup; i++) {
  logver = pasoE(&arr_arr[i], n_clases, n_clases_sup, i, p_c, p_cd, p_cx, logpcsum);
  logver_total += logver;
  ant_logver[i] = logver;
}

ant_error_val = 200;
error_val = 10;
ant_logver_total = −∞_d;

/* Dentro del bucle principal el orden será paso M—paso E */
salir = 0;
while (¬salir) {
  ant_logver_total = logver_total;
  logver_total = 0.0;
  suavizados = 0;

  if (flags & FL_VAL ∧ flags & FL_END_SMOOTH)
    intercambia_dmat2D(&p̃_cd, &ant_pcd);
  else if (flags & FL_VAL)
    intercambia_dmat2D(&p_cd, &ant_pcd);
```

```
    for (i=1; i≤n_clases_sup; i++) {
      /* Actualización de las probabilidades a priori p(c) = (Σ_n p(c|x_n)) / N */
      arr = &arr_arr[i];
      for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++) {
        suma = 0.0;
        for (x=0; x<arr→n_doc; x++)
          suma += p_cx[c][x];
        p_c[c] = suma / arr→n_doc;
      }

      /* Paso M
                          p_c = 1 / (Σ_n p(c|x_n) Σ_d x_nd) Σ_n p(c|x_n) x_n
      */
      for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++) {
        /* En p_cd[c][0] almacenaremos el denominador */
        for (d=0; d≤n_pal; d++)
          p_cd[c][d] = 0;
        for (x=0; x<arr→n_doc; x++) {
          doc = &arr→doc[x];
          suma = 0.0;
          for (d=0; d<doc→len; d++) {
            p_cd[c][pos_vd_float(&doc→v, d)] += p_cx[c][x] * val_vd_float(&doc→v, d);
            suma += val_vd_float(&doc→v, d);
          }
          p_cd[c][0] += suma * p_cx[c][x];
        }

        /* Suavizado (si corresponde) */

        if (flags & FL_INTER_SMOOTH)
          suavizado(p_cd, p̃_cd, β, par_smooth, c, n_clases, n_pal, flags);

        if (flags & FL_END_SMOOTH)
          p̃_cd[c][0] = p_cd[c][0];
        for (d=1; d≤n_pal; d++) {
          if (flags & FL_END_SMOOTH)
            p̃_cd[c][d] = p_cd[c][d]; /* Almacenamos las cuentas */
          p_cd[c][d] /= p_cd[c][0];
        }
      }
```

```
    /* Paso E. Si hemos suavizado tenemos que decidir si aceptamos el suavizado o no. */

    if (¬(flags & FL_INTER_SMOOTH)) {
      logver = pasoE(&arr_arr[i], n_clases, n_clases_sup, i, p_c, p_cd, p_cx, logpcsum);
      logver_total += logver;
      ant_logver[i] = logver;
    } else {
      logver = pasoE(&arr_arr[i], n_clases, n_clases_sup, i, p_c, p̃_cd, p_cx, logpcsum);
      if (logver > ant_logver[i]) {
        suavizados++;
        for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++)
          for (d=1; d≤n_pal; d++)
            p_cd[c][d] = p̃_cd[c][d];
      } else {
        logver = pasoE(&arr_arr[i], n_clases, n_clases_sup, i, p_c, p_cd, p_cx, logpcsum);
      }
      ant_logver[i] = logver;
      logver_total += logver;
    }
  }

  error_tr = 0;
  if (flags & FL_NO_SUP) {
    error_tr = calcula_error_nosup(&arr_arr[1],&arr_arr[1], p_c, p_cx, p_cd,
n_clases_sup_real,
                                   n_clases, flags, 0);
  } else {
    for (i=1; i≤n_clases_sup; i++)
      error_tr += calcula_error_sup(&arr_arr[i], n_clases, n_clases_sup, p_i, p_c, p_cd, flags,
0);
    error_tr /= n_clases_sup;
  }

  if (flags & FL_VAL) {
    ant_error_val = error_val;
    if (flags & FL_NO_SUP)
      error_val = calcula_error_nosup(&arr_arr[1], arr_val, p_c, p_cx, p_cd, n_clases_sup_real,
                                      n_clases, flags, 0);
    else
      error_val = calcula_error_sup(arr_val, n_clases, n_clases_sup, p_i, p_c, p_cd, flags, 0);
  } else if (flags & FL_FAKE_VAL) {
```

```
        if (flags & FL_NO_SUP)
          error_val = calcula_error_nosup(&arr_arr[1], arr_val, p_c, p_cx, p_cd, n_clases_sup_real,
                                          n_clases, flags, 0);
        else
          error_val = calcula_error_sup(arr_val, n_clases, n_clases_sup, p_i, p_c, p_cd, flags, 0);
      }
      printf("%d\t%g\t\t%g\t%g\t%d\n", it, logver_total, error_tr, error_val,
suavizados);
      fprintf(stderr, "%d\t%g\t\t%g\t%g\t%d\n", it, logver_total, error_tr, error_val,
suavizados);

      salir = (++it > max_iter);
      if (¬(flags & FL_FORCE_ITER)) {
        salir = salir ∨ (fabs((logver_total − ant_logver_total)/logver_total) < 1e−6);
        if (flags & FL_VAL)
          salir = salir ∨ error_val ≤ ant_error_val;
      }
    }

    if (flags & FL_VAL ∧ flags & FL_END_SMOOTH)
      intercambia_dmat2D(&p̃_cd, &ant_pcd);
    else if (flags & FL_VAL)
      intercambia_dmat2D(&p_cd, &ant_pcd);

    if (flags & FL_END_SMOOTH) {
      for (i=1; i≤n_clases_sup; i++) {
        for (c=no_superv(1,i); c≤no_superv(n_clases,i); c++) {
          suavizado(p̃_cd, p_cd, β, par_smooth, c, n_clases, n_pal, flags);
        }
      }
    }

    if (flags & FL_ESCRIBE)
      escribe_clasif(n_clases, n_pal, p_c, p_cd, "clasif.em");

    if (flags & FL_TEST) {
      if (flags & FL_NO_SUP) {
        /* Necesitamos calcular los nuevox p_cx. */
        pasoE(&arr_arr[1], n_clases, n_clases_sup, 1, p_c, p_cd, p_cx, logpcsum);
        calcula_error_nosup(&arr_arr[1],arr_te, p_c, p_cx, p_cd, n_clases_sup_real,
                            n_clases, flags, 1);
      } else
```

calcula_error_sup(*arr_te*, *n_clases*, *n_clases_sup*, p_i, p_c, p_{cd}, *flags*, 1);
}

libera2D(p_{cx});
if (*flags* & *FL_SMOOTH*)
libera2D($\tilde{p}_{cd}$);
if (*flags* & *FL_VAL*)
libera2D(*ant_pcd*);
free(*ant_logver*);
free(p_c);
free(*logpcsum*);
libera2D(p_{cd});
free(p_i);
}

C.2.6. Funciones de control del programa

ayuda()

Muestra las opciones del programa.

void ayuda(**char** ∗*nombre*)
{
fprintf(*stderr*,

```
"\n"
"Uso: %s -n <n_clases> [-c <n_clases_sup>] -tr <training_base> [opciones]\n"
"\n"
"Con:\n"
"\t-n, --nclases:\t\tnúmero de clases no supervisadas\n"
"\t-c, --clases-sup:\tindica aprendizaje supervisado con <n_clases_sup>\n"
"\t                 \tclases supervisadas\n"
"\t-tr, --training:\tconjunto de training o \"nombre base\" del conjunto.\n"
"\t                \tPara el caso de aprendizaje superviado los ficheros\n"
"\t                \tque se leerán serán nombre1, nombre2, ... nombreC\n"
"\n"
"Y las opciones:\n"
"\t-d, --dicc:\t\tEspecifica un diccionario\n"
"\t-te, --test:\t\tConjunto de test\n"
"\t-val, --validacion:\tConjunto de validación\n"
"\t-fval, --fake-val:\t\"Conjunto de validación\" que no detiene el algoritmo\n"
"\t-i, --iter:\t\tMáximo número de iteraciones\n"
"\t-fi, --forceiter:\t\tFuerza la ejecución de un número determinado de iteraciones\n"
"\t-norm:\t\t\tNormalización de la longiud de documentos\n"
"\t-h, --help:\t\tMuestra esta ayuda\n"
"\t-ri, --randini:\t\tInicialización aleatoria\n"
"\t-mi, --muestraini:\tInicialización por una muestra\n"
"\t--maxminini:\t\tInicialización por maxmin\n"
"\t-s, --seed:\t\tEspecifica una semilla (0 = time(NULL))\n"
"\n"
"\t-addeps, --laplace:\tSuavizado de Laplace\n"
```

```
        "\t-adb0,--unifbackoff:\tSuavizado␣uniform␣backoff\n"
        "\t-adb1,--unigbackoff:\tSuavizado␣unigram␣backoff\n"
        "\t-adi0,--unifinter:\tSuavizado␣uniform␣interpolation\n"
        "\t-adi1,--uniginter:\tSuavizado␣unigram␣interpolation\n"
        "\n"
        "␣␣Con␣una␣i␣inicial␣(ej.␣-iadb0)␣aplica␣suavizado␣\"intermedio\",␣es␣decir,\n"
        "␣␣en␣las␣iteraciones␣del␣EM.\n"
        "\n",
```

nombre);
}

argumentos()

Trata la línea de comandos.

t_flags argumentos(**int** *argc*, **char** ∗∗*argv*, **char** ∗∗*fname_dicc*,
char ∗∗*fname_tr*, **char** ∗∗*fname_te*, **char** ∗∗*fname_val*,
int ∗*n_clases*, **int** ∗*iter*, **int** ∗*n_clases_sup*,
double ∗*par_smooth*, **unsigned int** ∗*semilla*)
{
int *i*;
t_flags *flags*=0*x0*;

∗*iter* = *MAX_ITER*;
∗*fname_dicc* = *NULL*;
∗*n_clases* = −1;
∗*n_clases_sup* = −1;
∗*semilla* = 0;
for (*i*=1; *i*<*argc*; *i*++) {
if (¬strcmp("-d", *argv*[*i*]) ∨ ¬strcmp("--dicc", *argv*[*i*])) {
flags |= *FL_DICC*;
∗*fname_dicc* = *argv*[++*i*];
} **else if** (¬strcmp("-c", *argv*[*i*]) ∨ ¬strcmp("--clases-sup", *argv*[*i*])) {
∗*n_clases_sup* = atoi(*argv*[++*i*]);
} **else if** (¬strcmp("-tr", *argv*[*i*]) ∨ ¬strcmp("--training", *argv*[*i*])) {
∗*fname_tr* = *argv*[++*i*];
} **else if** (¬strcmp("-te", *argv*[*i*]) ∨ ¬strcmp("--test", *argv*[*i*])) {
flags |= *FL_TEST*;
∗*fname_te* = *argv*[++*i*];
} **else if** (¬strcmp("-val", *argv*[*i*]) ∨ ¬strcmp("--validacion", *argv*[*i*])) {
flags |= *FL_VAL*;
∗*fname_val* = *argv*[++*i*];
} **else if** (¬strcmp("-fval", *argv*[*i*]) ∨ ¬strcmp("--fake-validacion", *argv*[*i*])) {

```
    flags |= FL_FAKE_VAL;
    *fname_val = argv[++i];
  } else if (¬strcmp("-n", argv[i]) ∨ ¬strcmp("--nclases", argv[i])) {
    *n_clases = atoi(argv[++i]);
  } else if (¬strcmp("-i", argv[i]) ∨ ¬strcmp("--iter", argv[i])) {
    *iter = atoi(argv[++i]);
  } else if (¬strcmp("-fi", argv[i]) ∨ ¬strcmp("--forceiter", argv[i])) {
    flags |= FL_FORCE_ITER;
    *iter = atoi(argv[++i]);
  } else if (¬strcmp("-norm", argv[i])) {
    flags |= FL_NORM;
  } else if (¬strcmp("-ri", argv[i]) ∨ ¬strcmp("--randini", argv[i])) {
    flags |= FL_RAND_INI;
  } else if (¬strcmp("-mi", argv[i]) ∨ ¬strcmp("--muestraini", argv[i])) {
    flags |= FL_MUESTRA_INI;
  } else if (¬strcmp("--maxminini", argv[i])) {
    flags |= FL_MAXMIN_INI;
  } else if (¬strcmp("-s", argv[i]) ∨ ¬strcmp("--seed", argv[i])) {
    *semilla = atoi(argv[++i]);

/* Suavizado final */
  } else if (¬strcmp("-addeps", argv[i]) ∨ ¬strcmp("--laplace", argv[i])) {
    flags |= (FL_LAPLACE | FL_END_SMOOTH);
    *par_smooth = atof(argv[++i]);
  } else if (¬strcmp("-adb0", argv[i]) ∨ ¬strcmp("--unifbackoff", argv[i])) {
    flags |= (FL_BACKOFF | FL_UNIFORM | FL_END_SMOOTH);
    *par_smooth = atof(argv[++i]);
  } else if (¬strcmp("-adb1", argv[i]) ∨ ¬strcmp("--unigbackoff", argv[i])) {
    flags |= (FL_BACKOFF | FL_UNIGRAM | FL_END_SMOOTH);
    *par_smooth = atof(argv[++i]);
  } else if (¬strcmp("-adi0", argv[i]) ∨ ¬strcmp("--unifinter", argv[i])) {
    flags |= (FL_INTERPOLATION | FL_UNIFORM | FL_END_SMOOTH);
    *par_smooth = atof(argv[++i]);
  } else if (¬strcmp("-adi1", argv[i]) ∨ ¬strcmp("--uniginter", argv[i])) {
    flags |= (FL_INTERPOLATION | FL_UNIGRAM | FL_END_SMOOTH);
    *par_smooth = atof(argv[++i]);

    /* Suavizado intermedio */
  } else if (¬strcmp("-iaddeps", argv[i]) ∨ ¬strcmp("--laplace", argv[i])) {
    flags |= (FL_LAPLACE | FL_INTER_SMOOTH);
    *par_smooth = atof(argv[++i]);
```

```
    } else if (¬strcmp("-iadb0", argv[i]) ∨ ¬strcmp("--unifbackoff", argv[i])) {
      flags |= (FL_BACKOFF | FL_UNIFORM | FL_INTER_SMOOTH);
      *par_smooth = atof(argv[++i]);
    } else if (¬strcmp("-iadb1", argv[i]) ∨ ¬strcmp("--unigbackoff", argv[i])) {
      flags |= (FL_BACKOFF | FL_UNIGRAM | FL_INTER_SMOOTH);
      *par_smooth = atof(argv[++i]);
    } else if (¬strcmp("-iadi0", argv[i]) ∨ ¬strcmp("--unifinter", argv[i])) {
      flags |= (FL_INTERPOLATION | FL_UNIFORM | FL_INTER_SMOOTH);
      *par_smooth = atof(argv[++i]);
    } else if (¬strcmp("-iadi1", argv[i]) ∨ ¬strcmp("--uniginter", argv[i])) {
      flags |= (FL_INTERPOLATION | FL_UNIGRAM | FL_INTER_SMOOTH);
      *par_smooth = atof(argv[++i]);

    } else if (¬strcmp("-h", argv[i]) ∨ ¬strcmp("--help", argv[i])) {
      ayuda(argv[0]);
      exit(0);
    } else {
      fprintf(stderr, "Opción␣desconocida:␣%s\n", argv[i]);
      ayuda(argv[0]);
      exit(1);
    }
  }

  if (*n_clases == -1) {
    fprintf(stderr,
"Error:␣Hay␣que␣especificar␣un␣número␣de␣clases␣(opción␣-n).\n");
    exit(1);
  }
  if (*n_clases_sup == -1) {
    flags |= FL_NO_SUP;
    *n_clases_sup = 1;
  }
  if (¬(flags & (FL_MUESTRA_INI | FL_RAND_INI | FL_MAXMIN_INI)))
    flags |= FL_MAXMIN_INI;

  return(flags);
}
```

La función main()

```
int main(int argc, char **argv)
{
```

```
t_arr_doc *arr_tr, arr_te, arr_val;
char *fname_dicc, *fname_tr, *fname_te, *fname_val;
t_flags flags;
t_hash dicc, clases;
t_buffer buf;
int i, n_clases, iter, n_clases_sup, x, d;
FILE *fd;
double par_smooth;
float val;
t_doc *doc;
unsigned int semilla;

crea_buffer(&buf, TAM_BUFFER, MAX_LEN);
crea_hash(&dicc, TAM_HASH);
crea_hash(&clases, 40);

flags=argumentos(argc, argv, &fname_dicc, &fname_tr, &fname_te, &fname_val,
                 &n_clases, &iter, &n_clases_sup, &par_smooth, &semilla);
if (flags & FL_DICC)
  lee_dicc(&dicc, &buf, fname_dicc);
if (¬semilla)
  srand(time(NULL));
else
  srand(semilla);

mt(arr_tr = (t_arr_doc *) malloc((n_clases_sup+1)*sizeof(t_arr_doc)));
for (i=1; i≤n_clases_sup; i++) {
  if (¬(flags & FL_NO_SUP))
    sprintf(buf.pos_buf, "%s%d", fname_tr, i);
  else
    strcpy(buf.pos_buf, fname_tr);
  crea_arr_doc(&arr_tr[i], TAM_BUF_DOC);
  lee_arr_doc(&arr_tr[i], &dicc, ¬(flags & FL_DICC), &clases, &buf, buf.pos_buf);
  if (flags & FL_NORM) {
    for (x=0; x≤arr_tr[i].n_doc; x++) {
      doc = &arr_tr[i].doc[x];
      for (d=0; d<doc→len; d++) {
        val = val_vd_float(&doc→v, d);
        pon_val_vd_float(&doc→v, d, val/doc→len);
      }
    }
```

```
        }
      }

      if (flags & FL_TEST) {
        crea_arr_doc(&arr_te, TAM_BUF_DOC);
        lee_arr_doc(&arr_te, &dicc, 0, &clases, &buf, fname_te);
      }
      if (flags & (FL_VAL | FL_FAKE_VAL)) {
        crea_arr_doc(&arr_val, TAM_BUF_DOC);
        lee_arr_doc(&arr_val, &dicc, 0, &clases, &buf, fname_val);
      }

      em(arr_tr, n_clases, n_clases_sup, &clases, get_maxpos_hash(&dicc), iter, flags,
          (flags & FL_TEST) ? &arr_te : NULL, (flags & (FL_VAL | FL_FAKE_VAL)) ? &arr_val :
    NULL,
          par_smooth);
      if (flags & FL_ESCRIBE_DICC) {
        fd=fopen("dicc.em", "w");
        escribe_hash(&dicc, fd);
        fclose(fd);
      }

      return 0;
    }
```

APÉNDICE D

Resultados adicionales

En este apéndice presentamos resultados adicionales, no incluidos en el texto principal para no entorpecer la lectura del mismo, pero que son necesarios para un estudio más detallado de los resultados obtenidos.

D.1. Clasificador multinomial (Capítulo 3)

Las figuras D.1 y D.2 muestran las matrices de confusión obtenidas con el mejor clasificador para los corpus EuTran y WebKB[1], respectivamente.

	A	**F**	**J**	**P**
A	986	10	4	0
F	109	887	1	3
J	0	0	1000	0
P	0	0	0	1000

FIGURA D.1: Matriz de confusión para el corpus EuTrans utilizando un clasificador multinomial.

[1]Para el corpus WebKB se muestra la matriz de confusión conjunta de todas las universidades.

	course	**faculty**	**project**	**student**
course	222	9	4	9
faculty	0	116	13	24
project	2	9	66	9
student	16	54	14	474

FIGURA D.2: Matriz de confusión para el corpus WebKB utilizando un clasificador multinomial.

D.2. Aprendizaje supervisado de mixturas (Capítulo 5)

D.2.1. WebKB

En la figura D.3 se representa la influencia del parámetro de suavizado sobre cada una de las técnicas estudiadas (utilizando el número óptimo de componentes no supervisadas).

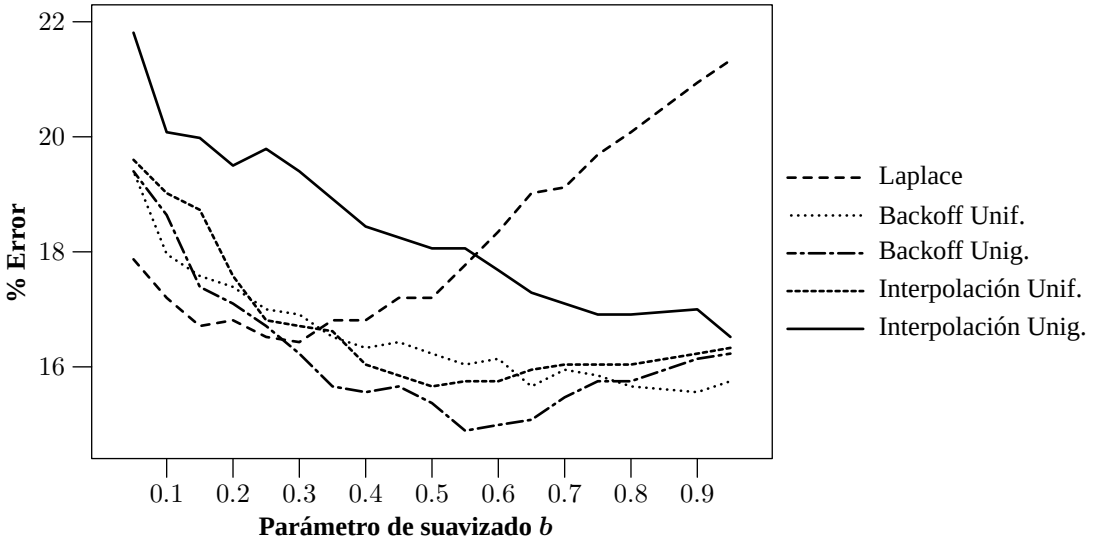

FIGURA D.3: Influencia del parámetro de suavizado b para el corpus WebKB utilizando aprendizaje supervisado de mixturas.

La figura D.4 muestra la matriz de confusión[2] obtenida para el corpus WebKB, utilizando los parámetros óptimos.

[2]Incluyendo las 5 repeticiones variando la semilla.

	course	**faculty**	**project**	**student**
course	1110	25	25	60
faculty	0	595	45	125
project	20	35	335	40
student	50	275	75	2390

FIGURA D.4: Matriz de confusión para el corpus WebKB utilizando aprendizaje supervisado de mixturas.

D.2.2. 20 newsgroups

La figura D.3 muestra la evolución del error variando el parámetro de suavizado sobre el corpus 20 newsgroups (utilizando el número óptimo de componentes no supervisadas) y la figurea D.3 la matriz de confusión obtenida utilizando suavizado backoff uniforme ($b = 0.35$) y dos componentes no supervisadas.

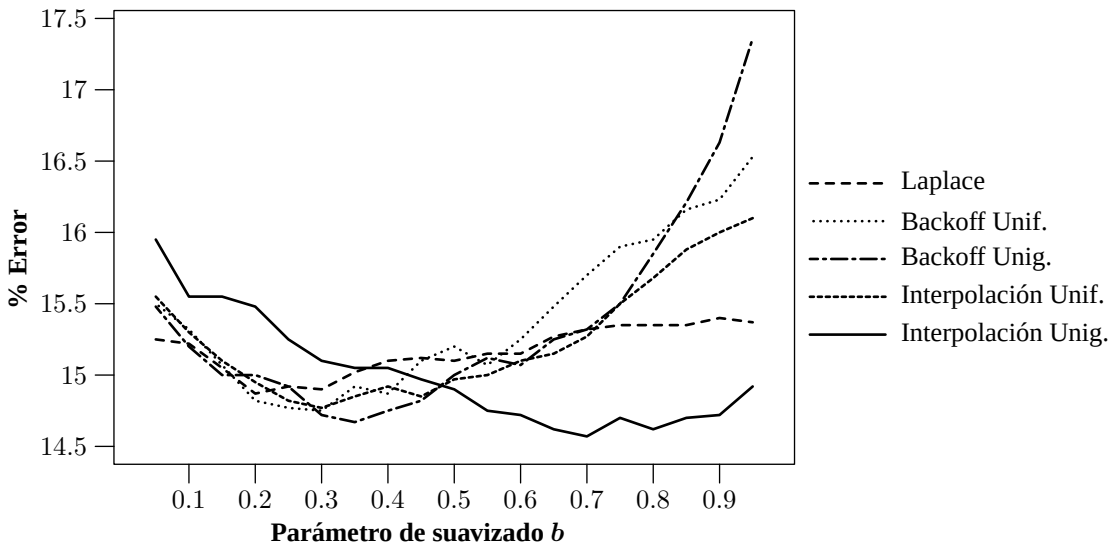

FIGURA D.5: Influencia del parámetro de suavizado b para el corpus WebKB utilizando aprendizaje supervisado de mixturas.

	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	785	0	0	0	0	0	5	0	10	0	0	0	0	0	5	25	0	0	5	165
comp.graphics	0	740	40	50	20	35	10	5	0	0	0	45	30	10	10	0	0	0	0	5
comp.os.ms-windows.misc	0	55	675	65	30	45	5	0	0	5	0	5	5	5	0	0	0	0	10	10
comp.sys.ibm.pc.hardware	0	15	65	775	55	0	40	0	0	0	5	0	40	0	5	0	0	0	0	0
comp.sys.mac.hardware	0	25	40	35	840	0	25	5	0	0	0	0	15	5	0	0	0	0	5	0
comp.windows.x	0	80	75	35	15	770	0	0	0	0	0	0	0	0	10	0	5	0	0	0
misc.forsale	0	5	5	55	10	0	840	10	10	10	5	0	35	10	0	0	0	0	0	5
rec.autos	0	5	5	0	0	0	10	965	5	0	0	0	0	0	0	0	0	0	10	0
rec.motorcycles	0	0	0	0	0	0	10	20	960	0	0	0	10	0	0	0	0	0	0	0
rec.sport.baseball	0	0	0	0	0	0	0	15	0	965	15	0	0	0	0	5	0	0	0	0
rec.sport.hockey	0	0	0	0	0	0	0	0	0	15	965	5	0	0	0	10	0	0	5	0
sci.crypt	0	10	5	5	0	5	20	0	0	0	0	910	10	5	0	0	0	0	5	0
sci.electronics	5	40	5	55	5	5	5	25	20	0	0	15	800	15	0	5	0	0	5	0
sci.med	0	10	5	5	0	0	0	5	0	0	10	0	15	900	20	0	0	0	5	20
sci.space	0	10	0	10	0	0	0	0	0	0	0	5	10	15	910	0	5	0	25	10
soc.religion.christian	10	5	5	5	0	0	0	0	0	0	5	0	0	0	5	920	0	0	5	25
talk.politics.guns	0	0	0	0	5	0	0	0	0	0	0	5	0	0	0	0	910	0	40	35
talk.politics.mideast	15	5	0	0	0	0	0	0	5	5	0	0	0	5	0	10	5	920	30	0
talk.politics.misc	5	0	0	0	0	0	0	0	0	0	0	0	0	5	10	0	115	60	710	95
talk.religion.misc	220	5	0	0	0	0	0	0	0	0	0	5	0	0	10	35	45	0	35	645

FIGURA D.6: Matriz de confusión para el corpus 20 newsgroups utilizando aprendizaje supervisado de mixturas.

D.3. Aprendizaje no supervisado (Capítulo 5)

D.3.1. EuTrans

La tabla D.1 muestra los mejores resultados obtenidos para el corpus EuTrans para el aprendizaje no supervisado y la figura D.7 la matriz de confusión utilizando los mejores parámetros.

Suavizado	**Nº de clases**	**Parámetro b**	**% Error**
Laplace	68	0.05	6.22 %
Backoff Unif.	68	0.65	6.08 %
Backoff Unig.	68	0.8	6.00 %
Interpolación Unif.	68	0.8	6.05 %
Interpolación Unig.	68	0.8	6.05 %

TABLA D.1: Mejores resultados obtenidos para el corpus EuTrans utilizando aprendizaje no supervisado.

	A	**F**	**J**	**P**
A	941	29	30	0
F	97	897	5	1
J	15	14	971	0
P	0	48	1	951

FIGURA D.7: Matriz de confusión para el corpus EuTrans utilizando aprendizaje no supervisado.

D.3.2. WebKB

La figura D.8 muestra la matriz de confusión obtenida para el corpus WebKB con los mejores parámetros para el aprendizaje no supervisado.

	course	**faculty**	**project**	**student**
course	149	8	0	87
faculty	1	60	6	86
project	0	9	7	70
student	26	35	3	494

FIGURA D.8: Matriz de confusión para el corpus WebKB utilizando aprendizaje no supervisado.

APÉNDICE E

Contenido del CD

Adjunto a esta memoria se entrega un CD, cuyos contenidos se detallan a continuación:

`mult/` En este directorio se encuentran los archivos fuente de los programas implementados (apéndices B y C). También se incluye un archivo `Makefile` para facilitar la compilación.

`corpora/` Este directorio contiene los tres corpora que se han utilizado a lo largo del proyecto. Para cada uno se incluyen archivos con la extensión `.tr`, que son los archivos preprocesados utilizados para entrenamiento, otros con la extensión `.te`, que son los empleados como conjunto de test y la versión original de cada corpus.

`rainbow/` En este directorio se incluye el programa `rainbow` de Andrew K. McCallum, necesario para llevar a cabo el preproceso de los corpora.

`memoria.pdf` Este mismo documento en formato pdf.

 Apéndice E. Contenido del CD

Bibliografía

[CPR00] MIGUEL Á. CARREIRA-PERPIÑÁN Y STEVE RENALS. Practical identifiability of finite mixtures of multivariate Bernoulli distributions. *Neural Computation* **12**(1), 141–152 (enero 2000).

[Deg88] MORRIS H. DEGROOT. "Probabilidad y estadística". Addison-Wesley Iberoamericana (1988).

[DH73] RICHARD O. DUDA Y PETER E. HART. "Pattern Recognition and Scene Analysis". John Wiley & Sons (1973).

[DHS01] RICHARD O. DUDA, PETER E. HART Y DAVID G. STORK. "Pattern Classification". John Wiley & Sons (2001).

[Gro] CMU TEXT LEARNING GROUP. World wide knowledge base (web→kb) project. `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/`.

[JN02] ALFONS JUAN Y HERMANN NEY. Reversing and Smoothing the Multinomial Naive Bayes Text Classifier. En "Proc. of the 2nd Int. Workshop on Pattern Recognition in Information Systems (PRIS 2002)", páginas 200–212, Alacant (Spain) (abril 2002).

[Jua00] ALFONS JUAN. "Optimización de prestaciones en técnicas de aprendizaje no supervisado y su aplicación al reconocimiento de formas". Tesis Doctoral, Universitat Politècnica de València (2000). Director: Dr. E. Vidal.

[JV02] ALFONS JUAN Y ENRIQUE VIDAL. On the use of Bernoulli mixture models for text classification. *Pattern Recognition* **35**(12), 2705–2710 (diciembre 2002).

[LJ02] JAVIER LAFUENTE Y ALFONS JUAN. Comparación de Codificaciones de Documentos para Clasificación con K Vecinos Más Próximos. En "Proc. of the I Jornadas de Tratamiento y Recuperación de Información (JOTRI)", páginas 37–44, València (Spain) (julio 2002).

[McC96] ANDREW KACHITES MCCALLUM. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow` (1996).

[MP00] GEOFFREY MCLACHLAN Y DAVID PEEL. "Finite Mixture Models". Wiley series in Probability and statistics (2000).

[NLM99] KAMAL NIGAM, JOHN LAFFERTY Y ANDREW MCCALLUM. Using maximum entropy for text classification. En "IJCAI-99 Workshop on Machine Learning for Information Filtering" (1999).

[NMTM98] KAMAL NIGAM, ANDREW K. MCCALLUM, SEBASTIAN THRUN Y TOM M. MITCHELL. Learning to classify text from labeled and unlabeled documents. En "Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence", páginas 792–799, Madison, US (1998). AAAI Press, Menlo Park, US.

[NMTM00] KAMAL NIGAM, ANDREW K. MCCALLUM, SEBASTIAN THRUN Y TOM M. MITCHELL. Text classification from labeled and unlabeled documents using EM. *Machine Learning* **39**(2/3), 103–134 (2000).

[TSM85] D. M. TITTERTON, A. F. M. SMITH Y U. E. MAKOV. "Statistical Analysis of Finite Mixture Distributions". John Wiley & Sons (1985).

[YP97] YIMING YANG Y JAN O. PEDERSEN. A comparative study on feature selection in text categorization. En DOUGLAS H. FISHER, editor, "Proceedings of ICML-97, 14th International Conference on Machine Learning", páginas 412–420, Nashville, US (1997). Morgan Kaufmann Publishers, San Francisco, US.