UNIVERSIDAD JAUME I
DPTO. DE INGENIERÍA Y CIENCIA DE LOS COMPUTADORES

Técnicas de agrupamiento bilingüe aplicadas a la inferencia de traductores

Tesis Doctoral
Presentada por Sergio Barrachina Mir
Dirigida por el Doctor D. Juan Miguel Vilar Torres

Castellón, marzo de 2003

Técnicas de agrupamiento bilingüe aplicadas a la inferencia de traductores

Sergio Barrachina Mir

Trabajo realizado bajo la dirección del Doctor
D. Juan Miguel Vilar Torres
y presentado en la Universidad Jaume I
para optar al grado de Doctor en Ingeniería Informática

Castellón, marzo de 2003

Este trabajo ha sido parcialmente realizado dentro de los proyectos TransType 2 (proyecto IST-2001-32091), subvencionado por la Unión Europea; SIEsTA (proyecto P1·1B2002-15), subvencionado por BanCaja; NeuroTrad (proyecto P1A99-10), subvencionado por BanCaja; EuTrans (proyecto ESPRIT n. 30268), subvencionado por la Unión Europea y ExTra (TIC97-0745-C02-01), subvencionado por la Comisión Interministerial de Ciencia y Tecnología.

Capítulo 3

Agrupamiento monolingüe

3.1. Introducción

Como se ha visto en el capítulo anterior, el objetivo de un modelo estocástico de lenguaje es el de proporcionar una estimación de la probabilidad $\mathrm{P}(w_1 \dots w_N)$ de una secuencia de palabras $w_1 \dots w_N$. Esta probabilidad conjunta se calcula habitualmente como el producto de la secuencia de probabilidades condicionales $\mathrm{P}(w_i \mid w_1 \dots w_{i-1})$. Dicho producto, en el caso de utilizar un modelo de lenguaje de bigramas, queda reducido a $\prod_{i=1}^{N} \mathrm{P}(w_i \mid w_{i-1})$.

Uno de los principales problemas que presenta la estimación de un modelo de lenguaje es la escasez de los datos disponibles para su entrenamiento. Esto es así aún para el modelo relativamente sencillo de bigramas. Existen tantas posibles combinaciones de pares de palabras que es imposible observarlas todas el suficiente número de veces, por muy grande que sea el número de ejemplos del que dispongamos.

El agrupamiento de aquellas palabras que son similares entre sí es una forma de combatir la escasez de datos de entrenamiento disponibles. Si fuéramos capaces de agrupar satisfactoriamente palabras similares, podríamos realizar predicciones más razonables de aquellas secuencias de palabras que no se han visto asumiendo que son similares a otras ya vistas [BDd+92].

Es decir, un modelo de lenguaje puede utilizar las relaciones existentes entre grupos de palabras o categorías en lugar de basarse en las relaciones entre palabras individuales. Este enfoque presenta las siguientes ventajas [Nie97]:

- Los modelos basados en categorías comparten estadísticas entre palabras de la misma categoría y, por lo tanto, pueden generalizarlas a pautas de

palabras no encontradas en el corpus de entrenamiento. Esta habilidad de procesar los eventos no vistos de forma sensata mejora la *robustez* del modelo de lenguaje.

- Agrupar palabras en categorías puede reducir el número de contextos que debe considerar un modelo y, de ese modo, hacer frente a la escasez de datos del conjunto de entrenamiento.

- Además, la reducción en el número de contextos conlleva un modelo con menos parámetros y, por lo tanto, más compacto. Este modelo poseerá unos requerimientos de almacenamiento más modestos; lo que desde un punto de vista práctico es importante.

Por medio del agrupamiento se intenta aprovechar el hecho de que algunas palabras son similares entre sí, ya sea por su significado o por su función sintáctica. Por ejemplo, no nos resultaría sorprendente constatar que la distribución de probabilidad de las palabras próximas a *sábado* fuera similar a la de las palabras próximas a *martes*. Naturalmente, dichas distribuciones no tienen por qué ser idénticas: por ejemplo, difícilmente oiremos la frase *¡Gracias a Dios que ya es martes!* o a alguien preocuparse porque mañana sea *sábado 13* (mientras que si alguien es supersticioso puede preocuparle que mañana sea martes 13). Si hemos agrupado los días de la semana en una clase, podríamos suponer que en la mayor parte de contextos en los que aparece uno de los días de la semana podría aparecer cualquiera de los otros.

Es decir, el agrupamiento nos ayuda a generalizar; podríamos considerarlo como una forma de aprendizaje. Agrupamos palabras en clases y generalizamos lo que sabemos de algunos de los miembros de una clase a los restantes.

En este capítulo presentamos varios algoritmos que agrupan de forma automática palabras en clases. La siguiente sección muestra algunos conceptos generales de agrupamiento, así como los tipos de agrupamientos existentes y los algoritmos de agrupamiento monolingüe en los que se basan los propuestos en esta tesis. La sección 3.3 presenta la función que se quiere optimizar por medio del agrupamiento y que será utilizada por el algoritmo iterativo descrito en la sección 3.4. La sección 3.5 muestra un algoritmo idéntico al anterior pero en el que la función objetivo se obtiene utilizando un caso particular de validación cruzada. Por último, la sección 3.6 muestra un algoritmo que utiliza cualquiera de los dos anteriores para la obtención de un agrupamiento de forma incremental.

Las técnicas de agrupamiento descritas en este capítulo contemplan únicamente el caso de las probabilidades condicionales de bigramas $\mathrm{P}(w_i \mid w_{i-1})$; sin embargo, los conceptos presentados pueden ser fácilmente extendidos a trigramas o n-gramas en general.

3.2. Conceptos generales de agrupamiento

A lo largo de los años se han propuesto varias definiciones posibles del término agrupamiento (*clustering*); todas ellas generalmente basadas en la definición del término grupo o clase (*cluster*). Sin embargo, muchas de estas definiciones utilizan conceptos definidos pobremente tales como: «parecidos», «similares»... o por el contrario se refieren a grupos de un tipo determinado por lo que pierden su generalidad. Es más, muchas de las definiciones existentes son vagas y de tipo circular. Esto nos da una idea de la dificultad de encontrar una definición universalmente aceptada del término agrupamiento. La definición que hemos escogido, propuesta en [TK99, pp. 356–357], y que se presenta a continuación, responde al concepto de agrupamiento utilizado en esta tesis.

Sea X nuestro conjunto de datos,

$$X = \{x_1, x_2, \ldots, x_N\}\,. \tag{3.1}$$

Se define un m-agrupamiento de X, al que llamamos $\mathcal{G}$, como una partición de X en m conjuntos (*grupos* o *clases*), $c_1, \ldots, c_m$, de forma que se cumplan las siguientes condiciones:

- $c_i \neq \emptyset,\ i = 1, \ldots, m$,
- $\bigcup_{i=1}^{m} c_i = X$,
- $c_i \cap c_j = \emptyset,\ i \neq j,\ i,\ j = 1, \ldots, m$.

Además, buscaremos que los elementos de un grupo c_i sean «más similares» entre sí y «menos similares» a los elementos de los otros grupos. La cuantificación de los términos «similar» y «diferente» dependerá del tipo de grupos del que se hable.

Es conveniente notar que, de acuerdo a la anterior definición de agrupamiento, cada elemento pertenece a un sólo grupo. Este tipo de agrupamiento recibe el nombre de agrupamiento nítido[1] (*hard clustering* o *crisp clustering*).

Si quisiéramos encontrar el mejor agrupamiento posible, parece claro que, disponiendo del tiempo y recursos necesarios, la mejor forma de asignar cada elemento x_i a su grupo sería la de identificar todas las particiones posibles y seleccionar la más «razonable» de acuerdo a un criterio prefijado. Sin embargo, esta aproximación no es factible ni siquiera para un número moderado de elementos. Esto es debido a que el número de particiones distintas de N elementos en m grupos, $S(N, m)$, es [TK99, pp. 384]:

$$S(N, m) = \frac{1}{m!} \sum_{i=0}^{m} (-1)^{m-i} \left(\begin{array}{c} m \\ i \end{array} \right) i^N. \tag{3.2}$$

[1] Utilizamos el término *nítido* por oposición al término *difuso* y por ser traducción tanto de *hard* como de *crisp*; generalmente en castellano cuando se habla de agrupamiento se sobreentiende *hard-clustering* y cuando se quiere indicar *fuzzy clustering* se acompaña el término agrupamiento del adjetivo *difuso*. En [TK99, p. 357] se puede encontrar una definición de agrupamiento en términos de conjuntos difusos (*fuzzy sets*).

Dando valores a N y a m podemos obtener los siguientes resultados: $S(15, 3) = 2.375.101$, $S(20, 4) = 45.232.115.901$. Es decir, incluso para un número reducido de elementos, 15 y 20, agrupados en pocos grupos, 3 y 4, el número de particiones posibles es muy alto. Si aplicamos (3.2) para el número de elementos que querríamos agrupar en el caso de agrupamientos de palabras e intentamos agrupar 500 palabras (un vocabulario reducido) en tan sólo 2 grupos, el número de particiones que tendríamos que evaluar sería $S(500, 2) \simeq 10^{150}$. (Si la evaluación de cada posible agrupamiento llevara tan sólo 10^{-20} segundos, un programa que evaluara todas las posibles agrupaciones de 500 palabras en 2 grupos tardaría más de 10^{122} años.)

Como se puede ver, la búsqueda de una solución óptima explorando todas las posibles agrupaciones es computacionalmente inviable. De hecho, incluso el problema más reducido de encontrar el agrupamiento óptimo por medio de una enumeración completa pero no explícita utilizando técnicas de ramificación y poda o programación dinámica constituyen problemas NP-duros [Bru78, JC99].

Vista la imposibilidad de evaluar todas las posibles particiones para encontrar el agrupamiento óptimo, los algoritmos de agrupamiento generalmente utilizados proporcionan agrupamientos razonables, subóptimos, considerando sólo una pequeña fracción del conjunto de todas las particiones posibles de X. Puesto que la solución no es necesariamente la óptima, los resultados obtenidos dependerán de cuál sea el criterio utilizado para la selección de este subconjunto.

Aunque existen una gran cantidad de algoritmos de agrupamiento, éstos pueden clasificarse en unos pocos tipos. Podemos distinguir en primer lugar, entre agrupamientos jerárquicos y no jerárquicos o planos dependiendo del tipo de estructura del agrupamiento generado [MS00]. Un agrupamiento jerárquico se puede ver como un árbol donde cada nodo representa una subclase de su nodo padre. Las hojas del árbol son los objetos del conjunto a agrupar (en nuestro caso las palabras). Cada nodo representa un grupo formado por todos los objetos de sus descendientes. Dichas estructuras se representan generalmente utilizando dendogramas. A modo de ejemplo, la figura 3.1 representa mediante un dendograma el agrupamiento jerárquico de 22 palabras inglesas frecuentes.

Los algoritmos jerárquicos pueden subdividirse a su vez en [TK99, pág. 385]:

- Aglomerativos. Producen una secuencia de agrupamientos con un número cada vez menor de grupos. El agrupamiento producido a cada paso se consigue juntando en un grupo, dos de los grupos presentes en el paso anterior.

- Divisivos. Actúan en la dirección contraria a los anteriores; esto es, producen una secuencia de agrupamientos con un número creciente de grupos a cada paso. El agrupamiento producido a cada paso es el resultado de dividir en dos, uno de los grupos presentes en el paso anterior.

Por otro lado, los agrupamientos planos están formado por un número fijo de clases sin relación alguna entre ellas. Muchos de los algoritmos que generan

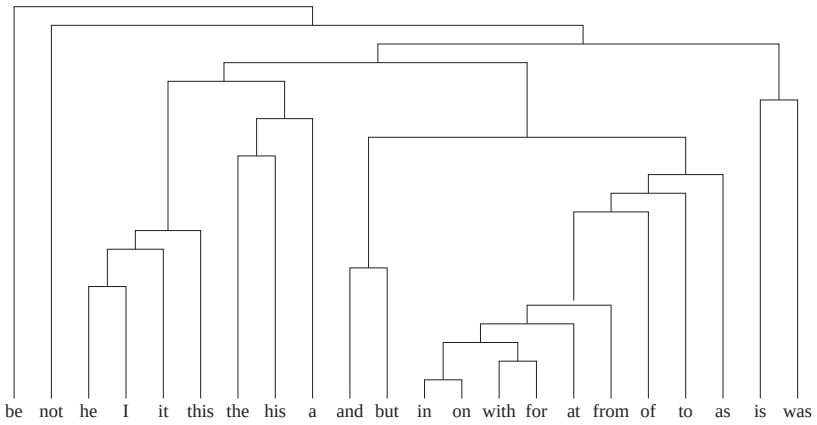

Figura 3.1: Dendograma que representa el agrupamiento jerárquico de 22 palabras inglesas frecuentes. Fuente: [MS00, pág. 496].

agrupamientos planos son *iterativos*. Comienzan con una distribución inicial de elementos en grupos y mejoran el agrupamiento mediante la recolocación iterativa de dichos elementos.

Otra distinción importante entre algoritmos de agrupamiento es si, como se ha visto antes, realizan un agrupamiento nítido o un agrupamiento difuso. En el agrupamiento nítido, cada objeto es asignado a un único grupo. Mientras que en el agrupamiento difuso se permite un cierto grado en la relación de pertenencia. Cada objeto puede pertenecer a más de un grupo. Dicha relación de pertenencia, puede expresarse en un marco probabilístico de la siguiente forma: para cada objeto x, se define una distribución de probabilidad $\mathrm{P}(\cdot \mid x)$ sobre los grupos, de tal forma que $\mathrm{P}(c \mid x)$ es la probabilidad de que x sea un miembro de c.

A pesar de su nombre, muchos de los algoritmos difusos asumen que un objeto pertenece en realidad a un sólo grupo, diferenciándose de los algoritmos nítidos en que existe una incertidumbre sobre cuál de los grupos posibles de un elemento es el correcto. También existen algoritmos difusos que sí que realizan asignación múltiple y donde cada elemento pertenece realmente a varios grupos. Dichos modelos reciben el nombre de agrupamientos disyuntivos [MS00].

Entre los agrupamientos jerárquicos, predominan los que asignan cada elemento a un único grupo. Entre los agrupamientos planos, los dos tipos de asignación son habituales.

Las principales características que presentan los algoritmos de agrupamiento jerárquicos y planos se muestran en el cuadro 3.1.

En cuanto al agrupamiento aplicado al campo del modelado de lenguaje natural, dos de los métodos de agrupamiento más conocidos son los propuestos por Brown et al. [BDd$^+$92]. En dicho trabajo se asume un agrupamiento nítido

Agrupamiento jerárquico:	Agrupamiento plano:
▪ Preferible para realizar análisis detallado de los datos. ▪ Proporciona mayor información que los agrupamientos planos. ▪ No existe un algoritmo mejor (varios algoritmos son óptimos para sus respectivas aplicaciones). ▪ Menos eficiente que el agrupamiento plano.	▪ Preferible si la eficiencia es importante o cuando los conjuntos de datos son muy grandes. ▪ El algoritmo de K-medianas es el método conceptualmente más simple y debería utilizarse en primer lugar sobre un conjunto de datos nuevos debido a que los resultados obtenidos son a menudo suficientes.

Cuadro 3.1: Resumen de las principales características de los algoritmos de agrupamiento.

en el que cada palabra w forma parte de sólo una clase $\mathcal{G}(w)$. Se propone la estimación basada en clases de la probabilidad de w_2 dada w_1 como:

$$\mathrm{P}(w_2 \mid w_1) = \mathrm{P}(w_2 \mid \mathcal{G}(w_2))\mathrm{P}(\mathcal{G}(w_2) \mid \mathcal{G}(w_1)). \tag{3.3}$$

Dado un corpus de entrenamiento y la función $\mathcal{G}$, tanto $\mathrm{P}(w \mid \mathcal{G}(w))$ como $\mathrm{P}(\mathcal{G}(w_2) \mid \mathcal{G}(w_1))$ pueden determinarse contando el número de veces que se dan en el entrenamiento determinados eventos. Por lo tanto, sólo es necesario encontrar la función $\mathcal{G}$. Brown et al. [BDd$^+$92] proponen encontrar esta función $\mathcal{G}$ de forma que la perplejidad del modelo de lenguaje basado en clases se reduzca; lo que equivale a maximizar la información mutua entre clases adyacentes $\mathrm{I}(c_1; c_2)$ (ver sección 3.3).

El primero de los algoritmos propuestos por Brown et al. [BDd$^+$92] es un algoritmo jerárquico aglomerativo en el que parten de una distribución inicial en la que hay tantas clases como palabras y van juntando dos clases cada vez hasta que se alcanza el número de clases deseado. En cada una de las iteraciones se prueban todos los posibles agrupamientos hasta que se encuentra aquel que produce el mayor aumento en $\mathrm{I}(c_1; c_2)$. Además, para compensar posibles agrupamientos prematuros de palabras en una misma clase, se desplazan palabras de unas clases a otras una vez se ha alcanzado el número deseado de clases. El algoritmo propuesto presenta un coste temporal $O(V^3)$, donde V es la talla del vocabulario.

Para disminuir dicho coste, Brown et al. [BDd$^+$92] presentan otro algoritmo que reduce la complejidad temporal a $O(V \cdot C^2)$, siendo C el número de clases deseadas. Este algoritmo ordena las palabras por su frecuencia y coloca las C primeras en una clase cada una. En cada iteración se añade la siguiente palabra más frecuente aún no agrupada como una nueva clase, buscando entonces qué dos clases es mejor unir. Cuando se ha realizado la fusión, el sistema vuelve a tener C clases. Pese a la reducción en coste temporal obtenida, es posible que este

heurístico acote tanto la búsqueda que muchos buenos agrupamientos no sean tenidos en cuenta [Lee97].

Otro de los métodos de agrupamiento monolingüe ampliamente citado, es el de Kneser y Ney [KN93]. En muchos aspectos, el trabajo de Kneser y Ney es bastante similar al de Brown et al. Utilizan el mismo modelo de probabilidad condicional basada en clases, y algunos de los heurísticos para acelerar los cálculos son también idénticos. Sin embargo, el criterio de optimización difiere, a pesar de que también deriva del principio de máxima-verosimilitud. La diferencia más importante es que en lugar de utilizar un algoritmo aglomerativo, Kneser y Ney [KN93] utilizan un algoritmo de agrupamiento plano que, por lo tanto, tiene desde el principio el número deseado de clases. La operación básica que realizan para buscar un mejor agrupamiento es el movimiento de una palabra desde su clase actual a otra. El coste temporal de dicho algoritmo para cada iteración es $O(N + V \cdot C^2)$, donde N es el número de palabras del corpus de entrenamiento, V la talla del vocabulario y C el número de clases deseadas.

Martin et al. [MLN95] mejoran el método propuesto por Kneser y Ney, organizando el corpus de entrenamiento de la siguiente forma: para cada par de palabras observadas, almacenan su cuenta. De esta forma, en lugar de visitar cada ocurrencia de la palabra w en el corpus de entrenamiento, se recopilan todos aquellos bigramas en los que aparece w. Asumiendo que dichas cuentas se almacenan en una matriz de acceso directo, obtienen una complejidad temporal para cada iteración de: $O(B + V \cdot C^2)$, donde B es el número de bigramas vistos, que habitualmente es bastante menor que el número de palabras de entrenamiento.

Para vocabularios grandes, el acceso directo se vuelve prohibitivo debido a los grandes requerimientos de memoria. Por ello, Martin et al. [MLN95] proponen utilizar listas y búsqueda binaria para almacenar las cuentas de bigramas. Puesto que por cada palabra hay de media B/V bigramas, su algoritmo presenta la siguiente complejidad temporal: $O(B \cdot \log(\frac{B}{V}) + V \cdot C^2)$.

Los algoritmos de agrupamiento monolingüe presentados en esta tesis están basados en los algoritmos de Brown et al. [BDd+92], Kneser y Ney [KN93] y Martin et al. [MLN95].

3.3. Función objetivo

El agrupamiento de palabras que queremos realizar tiene por objeto la obtención de un modelo de lenguaje mejor. Por lo tanto, es conveniente conocer la relación existente entre un agrupamiento concreto y el correspondiente modelo de lenguaje basado en clases. Conociendo esta relación, podremos buscar aquel agrupamiento que proporcione el mejor modelo de lenguaje basado en clases. En esta sección presentamos la función objetivo que utilizaremos en los algoritmos de agrupamiento propuestos.

Aplicando el modelo de bigramas, en el cual una palabra depende únicamente de la palabra anterior, podemos aproximar la entropía cruzada de un corpus

$\mathrm{L} = w_1 \dots w_N$ para una función $\mathcal{G}$ de asignación de palabras a grupos como:

$$\begin{aligned}
\mathrm{H}(\mathrm{L}, \mathcal{G}) &= -\frac{1}{N} \log \mathrm{P}_G(w_1 \dots w_N) \\
&\simeq -\frac{1}{N-1} \log \prod_{i=2}^{N} \mathrm{P}_G(w_i \mid w_{i-1}) \\
&\simeq -\frac{1}{N-1} \sum_{vw} \mathrm{n}(vw) \log \mathrm{P}_G(w \mid v). \qquad (3.4)
\end{aligned}$$

Si consideramos la función $\mathcal{G}$ como una función que, dada una palabra w, proporciona la clase c a la que ha sido asignada, no pudiendo pertenecer una palabra a más de una clase, podemos aproximar $\mathrm{P}_G(w \mid v)$ como:

$$\mathrm{P}_G(w \mid v) \simeq \mathrm{P}(w \mid c_2)\mathrm{P}(c_2 \mid c_1), \qquad (3.5)$$

donde $c_1 = \mathcal{G}(v)$ y $c_2 = \mathcal{G}(w)$. Ésta es la formula utilizada en Brown et al. [BDd+92] y ya presentada en (3.3).

Es decir, la probabilidad de que veamos la palabra w si hemos visto la palabra v viene dada mediante la probabilidad de w sabiendo su clase c_2 por la probabilidad de ver la clase c_2 una vez vista la clase c_1 (a la que pertenece la palabra v).

Sustituyendo (3.5) en (3.4), $\mathrm{H}(\mathrm{L}, \mathcal{G})$ queda:

$$\mathrm{H}(\mathrm{L}, \mathcal{G}) \simeq -\frac{1}{N-1} \sum_{vw} \mathrm{n}(vw) \log \mathrm{P}(w \mid c_2)\mathrm{P}(c_2 \mid c_1). \qquad (3.6)$$

Si realizamos las siguiente serie de operaciones [MS00, pp. 510–511]:

$$\begin{aligned}
\mathrm{H}(\mathrm{L}, \mathcal{G}) &\simeq -\Bigg[\sum_{uw} \frac{\mathrm{n}(uw)}{N-1} \Big(\log \mathrm{P}(w \mid c_2) + \log \mathrm{P}(c_2)\Big) \\
&\qquad + \sum_{uw} \frac{\mathrm{n}(uw)}{N-1} \Big(\log \mathrm{P}(c_2 \mid c_1) - \log \mathrm{P}(c_2)\Big)\Bigg] \\
&\simeq -\Bigg[\sum_{w} \frac{\sum_u \mathrm{n}(uw)}{N-1} \log \mathrm{P}(w \mid c_2)\mathrm{P}(c_2) \\
&\qquad + \sum_{c_1 c_2} \frac{\mathrm{n}(c_1 c_2)}{N-1} \log \frac{\mathrm{P}(c_2 \mid c_1)}{\mathrm{P}(c_2)}\Bigg] \\
&\simeq -\sum_{w} \mathrm{P}(w) \log \mathrm{P}(w) - \sum_{c_1 c_2} \mathrm{P}(c_1 c_2) \log \frac{\mathrm{P}(c_1 c_2)}{\mathrm{P}(c_1)\mathrm{P}(c_2)} \\
&\simeq \mathrm{H}(w) - \mathrm{I}(c_1; c_2), \qquad (3.7)
\end{aligned}$$

llegamos a que $\mathrm{H}(\mathrm{L}, \mathcal{G}) \simeq \mathrm{H}(w) - \mathrm{I}(c_1; c_2)$. Es decir, la entropía cruzada puede calcularse como la entropía del corpus dada la probabilidad de las palabras que lo forman, $\mathrm{H}(w)$, menos la información mutua entre clases adyacentes, $\mathrm{I}(c_1; c_2)$. Puesto que $\mathrm{H}(w)$ es independiente de la función $\mathcal{G}$ elegida, la función $\mathcal{G}_{opt}$ buscada será por tanto:

$$\mathcal{G}_{opt} = \arg\max_{\mathcal{G}} \mathrm{I}(c_1; c_2) = \arg\max_{\mathcal{G}} \sum_{c_1 c_2} \mathrm{P}(c_1 c_2) \log \frac{\mathrm{P}(c_1 c_2)}{\mathrm{P}(c_1)\mathrm{P}(c_2)}. \tag{3.8}$$

Para la obtención de un agrupamiento, basado en la optimización de la anterior función, se proponen en esta tesis los siguientes algoritmos:

- Algoritmo iterativo.
- Algoritmo iterativo dejando uno fuera (*leaving one out*).
- Algoritmo incremental.

El primero de ellos, el algoritmo iterativo, descrito en la siguiente sección, implementa un algoritmo similar al algoritmo de intercambio (*exchange algorithm*) utilizado en el agrupamiento convencional y busca mejorar $\mathrm{I}(c_1; c_2)$ sobre el corpus de entrenamiento.

El segundo, el algoritmo iterativo dejando uno fuera, descrito en la sección 3.5, es una variante del primero pero que utiliza la técnica de dejar uno fuera para el cálculo de $\mathrm{I}(c_1; c_2)$.

El último de ellos, el algoritmo incremental, descrito en la sección 3.6, comienza con una sola clase e incrementa una a una el número de clases hasta llegar al número deseado de clases; para cada número de clases utiliza alguno de los algoritmos anteriores para encontrar el mejor agrupamiento.

3.4. Algoritmo iterativo

El primero de los algoritmos presentados, al que llamamos algoritmo de agrupamiento iterativo, se basa en el algoritmo de agrupamiento de Kneser y Ney [KN93] junto con las mejoras de Martin et al. [MLN95] y una serie de consideraciones presentadas en esta sección. El criterio de optimización que hemos utilizado es el de la mejora de la perplejidad (ver sección 3.3).

Este algoritmo (ver algoritmo 3.1) genera una distribución inicial de palabras en clases y modifica esta distribución, moviendo iterativamente palabras de unas clases a otras, buscando aumentar la información mutua entre clases adyacentes, $\mathrm{I}(c_1; c_2)$.

La idea principal para cambiar la distribución es la siguiente: dada la función objetivo, $\mathrm{I}(c_1; c_2)$, que depende del agrupamiento, se calcula cómo se modificaría el valor de dicha función si una determinada palabra se moviera de la clase en la que se encuentra a cualquier otra; si alguno de los movimientos mejora $\mathrm{I}(c_1; c_2)$,

Algoritmo 3.1 Agrupamiento iterativo

Requiere: C, $maxIter$
  Inicializar $\mathcal{G}$
  repetir
    para cada w en orden descendiente de frecuencia **hacer**
      Mover tentativamente w a cada clase c
      Mover w a la clase c que más incremente $\mathrm{I}(c_1; c_2)$
    fin para
  hasta que se alcance $maxIter$ o no se haya movido ninguna palabra

se mueve definitivamente la palabra a aquella clase para la que se ha obtenido la mayor mejora. Este procedimiento se realiza para todas las palabras en un determinado orden y el proceso completo se repite hasta que ya no hay más movimientos o hasta que se alcanza un número prefijado de iteraciones.

Las palabras se evalúan en el siguiente orden: de las más frecuentes, es decir, las que aparecen más veces en el corpus, a las menos frecuentes. El utilizar dicho orden permite determinar primero la clase a la que pertenecen aquellas palabras de las que se posee una mayor evidencia en el corpus.

Además, otro aspecto a tener en cuenta es que el algoritmo propuesto generará un agrupamiento u otro dependiendo de cuál sea la distribución inicial de palabras en clases escogida. La distribución inicial propuesta en [BDd$^+$92] y en [MLN95] es la de colocar las palabras más frecuentes cada una en una clase y las restantes en la última clase. Esta distribución inicial, en la técnica propuesta en [BDd$^+$92] permite el agrupamiento de las palabras considerando en primer lugar a las palabras más frecuentes puesto que, como se comentó en la sección 3.2, esta técnica consiste en la asignación, en primer lugar, de las C palabras más frecuentes cada una a una clase y posteriormente en la creación de una nueva clase con la siguiente palabra y la unión de dos de estas clases en una sola, repitiendo estos dos pasos hasta que no queden más palabras. De esta forma, las palabras frecuentes son consideradas justo al comienzo del proceso y condicionan la distribución de palabras en clases.

Martin et al [MLN95] también proponen dicha distribución inicial. Sin embargo, en este caso el algoritmo impide que las palabras más frecuentes sean evaluadas al comienzo, ya que al estar cada una en una clase, no pueden abandonar dichas clases hasta que dejen de ser las únicas palabras en ellas. Una distribución inicial más adecuada, a nuestro juicio, sería la de colocar todas las palabras más frecuentes en una clase, y las $C - 1$ palabras menos frecuentes en las restantes $C - 1$ clases, ya que de esta forma, las palabras más frecuentes se mueven en primer lugar dando lugar a mejores agrupamientos [BV99c].

Partiendo de esta distribución inicial, las palabras se mueven de una clase a otra buscando mejorar la función objetivo $\mathrm{I}(c_1; c_2)$ (ver sección 3.3). Desde el punto de vista de la implementación del algoritmo conviene observar que

dicha función no tiene por qué recalcularse para cada movimiento si no que se puede calcular directamente el incremento o decremento, $\Delta \mathrm{I}(c_1; c_2)$, debido a cada movimiento de forma similar a la planteada en [MLN95].

Además, para reducir el tiempo de cómputo de los $\Delta \mathrm{I}(c_1; c_2)$ debidos al movimiento de una palabra w a cada una de las posibles clases $c_1, \ldots, c_C$ se han utilizado las siguientes técnicas:

- Se ha separado el cómputo del $\Delta \mathrm{I}(c_1; c_2)$ en dos partes: la debida a la extracción de w de c_w y la debida a la inserción de w en c_d. De esta forma, el $\Delta \mathrm{I}(c_1; c_2)$ debido a la extracción se calcula sólo una vez cuando se evalúan los posibles movimientos de la palabra w.
- Se limita el número de cuentas a recalcular obteniendo previamente aquellas clases que preceden y siguen a la palabra w. Como se verá más adelante estas clases son las únicas que intervienen en la modificación de $\mathrm{I}(c_1; c_2)$. Esto se hace una vez por palabra que se quiere evaluar.

Expresando $\mathrm{I}(c_1; c_2)$ en función de las cuentas sobre el corpus de entrenamiento, tenemos:

$$\mathrm{I}(c_1; c_2) \simeq \frac{1}{N-1} \left[\sum_{c_1 c_2} \mathrm{n}(c_1 c_2) \cdot \log \frac{\mathrm{n}(c_1 c_2)}{N-1} - 2 \sum_{c} \mathrm{n}(c) \cdot \log \frac{\mathrm{n}(c)}{N-1} \right]. \quad (3.9)$$

Podemos expresar la variación de $\mathrm{n}(c_1 c_2)$ y $\mathrm{n}(c)$ debida a la extracción de una palabra w de su clase c_w como:

$$\mathrm{n}'(c_1 c_2) = \begin{cases} \mathrm{n}(c_1 c_2) - \mathrm{n}(w c_2) & \text{si } c_1 = c_w \wedge c_2 \neq c_w \\ \mathrm{n}(c_1 c_2) - \mathrm{n}(c_1 w) & \text{si } c_1 \neq c_w \wedge c_2 = c_w \\ \mathrm{n}(c_1 c_2) - \mathrm{n}(c_w w) & \\ \quad - \mathrm{n}(w c_w) + \mathrm{n}(ww) & \text{si } c_1 = c_2 = c_w \\ \mathrm{n}(c_1 c_2) & \text{en otro caso} \end{cases} \quad (3.10)$$

$$\mathrm{n}'(c) = \begin{cases} \mathrm{n}(c) - \mathrm{n}(w) & \text{si } c = c_w \\ \mathrm{n}(c) & \text{en otro caso} \end{cases} \quad (3.11)$$

De igual forma, podemos calcular la variación de $\mathrm{n}(c_1 c_2)$ y $\mathrm{n}(c)$ si w se mueve a c_d como:

$$\mathrm{n} \gg (c_1 c_2) = \begin{cases} \mathrm{n}'(c_1 c_2) + \mathrm{n}(w c_2) & \text{si } c_1 = c_d \wedge c_2 \neq c_d \\ \mathrm{n}'(c_1 c_2) + \mathrm{n}(c_1 w) & \text{si } c_1 \neq c_d \wedge c_2 = c_d \\ \mathrm{n}'(c_1 c_2) + \mathrm{n}(ww) & \\ \quad + \mathrm{n}(w c_d) + \mathrm{n}(c_d w) & \text{si } c_1 = c_2 = c_d \\ \mathrm{n}'(c_1 c_2) & \text{en otro caso} \end{cases} \quad (3.12)$$

$$\mathrm{n} \gg (c) = \begin{cases} \mathrm{n}'(c) + \mathrm{n}(w) & \text{si } c = c_d \\ \mathrm{n}'(c) & \text{en otro caso} \end{cases} \quad (3.13)$$

En función de dichas cuentas, podemos calcular el $\Delta \mathrm{I}(c_1; c_2)$ debido a la extracción de la palabra w de la clase c_w como:

$$\begin{aligned}\Delta \mathrm{I}(c_1; c_2)\,|_{ext} \;\simeq\; & \frac{1}{N-1}\Bigg[\mathrm{n}'(c_w, c_w)\cdot \log\frac{\mathrm{n}'(c_w, c_w)}{N-1} \\ & -\mathrm{n}(c_w, c_w)\cdot \log\frac{\mathrm{n}(c_w, c_w)}{N-1} \\ & +\sum_{c: c\neq c_w}\left(\mathrm{n}'(c, c_w)\cdot \log\frac{\mathrm{n}'(c, c_w)}{N-1} - \mathrm{n}(c, c_w)\cdot \log\frac{\mathrm{n}(c, c_w)}{N-1}\right) \\ & +\sum_{c: c\neq c_w}\left(\mathrm{n}'(c_w, c)\cdot \log\frac{\mathrm{n}'(c_w, c)}{N-1} - \mathrm{n}(c_w, c)\cdot \log\frac{\mathrm{n}(c_w, c)}{N-1}\right) \\ & -2\left(\mathrm{n}'(c_w)\cdot \log\frac{\mathrm{n}'(c_w)}{N-1} - \mathrm{n}(c_w)\cdot \log\frac{\mathrm{n}(c_w)}{N-1}\right)\Bigg]. \end{aligned} \tag{3.14}$$

Por otro lado, podemos calcular el $\Delta \mathrm{I}(c_1; c_2)$ debido a la inserción de la palabra w en la clase c_d como:

$$\begin{aligned}\Delta \mathrm{I}(c_1; c_2)\,|_{ins} \;\simeq\; & \frac{1}{N-1}\Bigg[\mathrm{n} \gg (c_d, c_d)\cdot \log\frac{\mathrm{n} \gg (c_d, c_d)}{N-1} \\ & -\mathrm{n}(c_d, c_d)\cdot \log\frac{\mathrm{n}(c_d, c_d)}{N-1} \\ & +\sum_{c: c\neq c_d}\left(\mathrm{n} \gg (c, c_d)\cdot \log\frac{\mathrm{n} \gg (c, c_d)}{N-1} - \mathrm{n}'(c, c_d)\cdot \log\frac{\mathrm{n}'(c, c_d)}{N-1}\right) \\ & +\sum_{c: c\neq c_d}\left(\mathrm{n} \gg (c_d, c)\cdot \log\frac{\mathrm{n} \gg (c_d, c)}{N-1} - \mathrm{n}'(c_d, c)\cdot \log\frac{\mathrm{n}'(c_d, c)}{N-1}\right) \\ & -2\left(\mathrm{n} \gg (c_d)\cdot \log\frac{\mathrm{n} \gg (c_d)}{N-1} - \mathrm{n}'(c_d)\cdot \log\frac{\mathrm{n}'(c_d)}{N-1}\right)\Bigg]. \end{aligned} \tag{3.15}$$

Finalmente, obtenemos el $\Delta \mathrm{I}(c_1; c_2)$ debido al movimiento de la palabra w de la clase c_w a la clase c_d como:

$$\Delta \mathrm{I}(c_1; c_2) = \Delta \mathrm{I}(c_1; c_2)\,|_{ext} + \Delta \mathrm{I}(c_1; c_2)\,|_{ins}\ .$$

Si observamos las ecuaciones 3.10 y 3.12, podemos ver como efectivamente sólo hay variaciones en las cuentas en aquellos casos en los que o $\mathrm{n}(c, w) \neq 0$ o $\mathrm{n}(w, c) \neq 0$; es decir, en aquellos casos en los que una clase antecede o sigue a la palabra que se está evaluando. Por lo tanto, si cuando queremos evaluar el $\Delta \mathrm{I}(c_1; c_2)$ debido al movimiento de una palabra w obtenemos previamente las clases que la anteceden y la siguen, sólo tendremos que realizar las anteriores operaciones sobre estas clases. Ésta es la segunda de las técnicas de optimización propuestas anteriormente.

3.5. Algoritmo iterativo dejando uno fuera

Si observamos cómo varía con el número de clases la entropía medida sobre un conjunto de test, de un modelo de lenguaje basado en bigramas de clases, podemos comprobar como ésta inicialmente decae hasta alcanzar un mínimo, y como luego vuelve a ascender hasta llegar al valor de la entropía de un modelo de bigramas (cuando el número de clases es igual al del vocabulario). Este comportamiento se explica mediante las siguientes fases [Nie97]:

- Sobregeneralización. Cuando el número de clases es reducido, el modelo de lenguaje apenas puede distinguir entre secuencias de historias que realmente son diferentes entre sí.

- Compensación. Cuando el número de clases aumenta, el grado de generalización del modelo se reduce y la habilidad para discriminar entre diferentes historias mejora.

- Sobreentrenamiento. Cuando el número de clases es demasiado grande, el modelo de lenguaje comienza a reflejar las peculiaridades del conjunto de entrenamiento, por lo que es de esperar que se comporte peor sobre el conjunto de test.

Cuando la entropía se mide sobre el mismo conjunto de entrenamiento sobre el que se estiman las distintas probabilidades —y no sobre el conjunto de test—, la variación de ésta con el número de clases no sigue la pauta anterior. Puesto que los estimadores por máxima-verosimilitud de las probabilidades son las frecuencias relativas de los bigramas en el conjunto de entrenamiento, la entropía de un modelo de bigramas de clases medida sobre este conjunto de entrenamiento mejora conforme aumenta el número de clases, hasta que el número de clases es igual al del vocabulario; es decir, hasta que el modelo basado en clases equivale al modelo de bigramas. Por lo tanto, la entropía medida sobre el conjunto de entrenamiento es una función decreciente con el número de clases y no puede utilizarse para determinar el número óptimo de clases.

La figura 3.2 muestra la variación con el número de clases de las perplejidades medidas sobre el conjunto de test y sobre el conjunto de entrenamiento. Para la realización de este experimento se ha utilizado la parte en castellano del corpus castellano–inglés EuTrans I (ver capítulo 6) y los distintos agrupamientos se han obtenido con el algoritmo iterativo descrito en la sección anterior. En la figura se han representado: la perplejidad medida sobre el test para un modelo de bigramas y para modelos de bigramas de clases con distintos números de clases; y la perplejidad medida sobre el entrenamiento para un modelo de bigramas y para modelos de bigramas de clases con distintos números de clases. Si se observa la variación de la perplejidad medida sobre el test para los modelos de bigramas de clases para los distintos números de clases, se puede apreciar como ésta efectivamente disminuye al principio conforme el número de clases aumenta

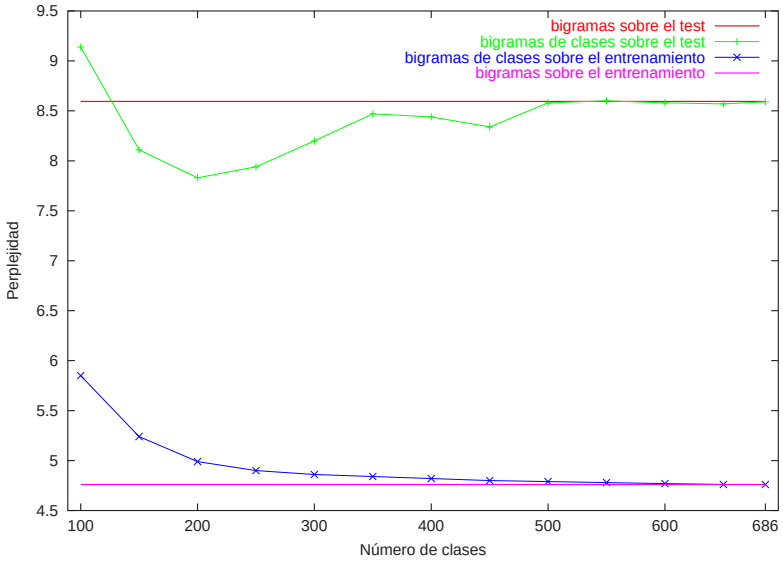

Figura 3.2: Perplejidad medida sobre el test y el entrenamiento para bigramas y bigramas de clases utilizando el algoritmo iterativo.

para después comenzar a ascender. Además, también se puede observar como la perplejidad medida sobre el conjunto de entrenamiento para los modelos de bigramas de clases disminuye conforme el número de clases aumenta y alcanza su valor mínimo cuando el número de clases es igual al del vocabulario del conjunto de entrenamiento (686 en el ejemplo).

Para evitar el sobreentrenamiento, se suele utilizar una técnica consistente en la división del corpus de entrenamiento en dos partes: a una de estas partes se le denomina la parte retenida (*retained part*) y a la otra, generalmente más pequeña, la parte separada (*held-out part*). Las estimaciones iniciales se llevan a cabo realizando las cuentas sobre la parte retenida y posteriormente se refinan sobre la parte separada. El coste que conlleva la utilización de esta técnica es la reducción de datos disponibles, siempre escasos, de entrenamiento, por lo que las estimaciones obtenidas pueden ser menos fiables [MS00].

Para solucionar el problema de la disminución de datos de entrenamiento, se aplican otros esquemas más eficientes en los que cada parte de los datos de entrenamiento se utiliza tanto para el entrenamiento inicial como para el posterior refinamiento. Por regla general, dichos métodos reciben en estadística el nombre de métodos de validación cruzada (*cross validation methods*).

El método de *dejar uno fuera* (*leaving-one-out method*) es un caso particular de validación cruzada. Este método divide el corpus de entrenamiento en una

parte retenida con $N-1$ muestras y en una parte separada con la muestra restante. El proceso se repite N veces de tal forma que se tienen en consideración todas las N particiones posibles con 1 muestra separada. La ventaja básica de esta aproximación es que todas las muestras son usadas tanto en la parte retenida como en la parte separada y por lo tanto se consigue aprovechar al máximo el corpus de entrenamiento disponible [KN93].

Utilizando este método para calcular la función objetivo en la que se basa el agrupamiento se puede evitar el sobreentrenamiento y obtener una estimación del número óptimo de clases.

Si llamamos T_i al corpus de entrenamiento sin el evento $w_{i-1}w_i$, la entropía cruzada del lenguaje según el modelo de lenguaje basado en clases (ecuación 3.4) se puede reescribir:

$$\mathrm{H}_{lo}(\mathrm{L},\mathcal{G}) \simeq -\frac{1}{N-1}\log\prod_{i=2}^{N}\mathrm{P}_{G,T_i}(w_i \mid w_{i-1}). \tag{3.16}$$

Si llamamos T_{vw} al corpus de entrenamiento sin uno de los eventos vw, tenemos:

$$\mathrm{H}_{lo}(\mathrm{L},\mathcal{G}) \simeq -\sum_{vw}\frac{\mathrm{n}(vw)}{N-1}\log\mathrm{P}_{G,T_{vw}}(c_2 \mid c_1)\mathrm{P}_{G,T_{vw}}(w \mid c_2), \tag{3.17}$$

donde, por simplificar la notación posterior, hemos denominado c_2 a $\mathcal{G}(w)$ y c_1 a $\mathcal{G}(v)$.

Tras realizar la serie de operaciones vistas en la sección 3.3, la parte de la ecuación 3.17 que depende del agrupamiento realizado queda:

$$\mathrm{I}_{lo}(c_1;c_2) \simeq \sum_{c_1c_2}\frac{\mathrm{n}(c_1c_2)}{N-1}\log\mathrm{P}_{G,T_{c_1c_2}}(c_1c_2) - 2\sum_{c}\frac{\mathrm{n}(c)}{N-1}\log\mathrm{P}_{G,T_c}(c), \tag{3.18}$$

donde abusando de la notación se ha denominado $T_{c_1c_2}$ al corpus de entrenamiento sin un evento vw cualquiera tal que $\mathcal{G}(v)=c_1 \wedge \mathcal{G}(w)=c_2$ y T_c al corpus de entrenamiento sin un evento vw cualquiera tal que $\mathcal{G}(v)=c \vee \mathcal{G}(w)=c$.

De ahora en adelante, se utilizará $\mathrm{P}_{G,T}$ para representar la probabilidad, dada $\mathcal{G}$, estimada sobre el corpus de entrenamiento del que se ha retirado un determinado evento vw.

Es decir, para cada evento vw presente en el corpus de entrenamiento queremos estimar $\mathrm{P}_{G,T}(c_1c_2)$ utilizando un corpus del que se ha eliminado dicho evento. Por lo tanto, es necesario considerar la posibilidad de que haya eventos no vistos en la parte retenida del corpus de entrenamiento a los que hay que asignar una probabilidad no nula. Para ello, siguiendo a Kneser y Ney [KN93], se utiliza el método de suavizado de descuento absoluto. Dicho método descuenta una cantidad $b<1$ a cada cuenta y reparte la masa de probabilidad ganada entre los eventos no vistos. Utilizando $n_{+,T}$ para representar el número de pares

c_1c_2 que se han visto en la parte retenida y $n_{0,T}$ para representar el número de los que no, se tiene que:

$$\mathrm{P}_{G,T}(c_1c_2) = \begin{cases} \dfrac{\mathrm{n}_T(c_1c_2) - b}{N_T} & \text{si } \mathrm{n}_T(c_1c_2) > 0 \\ \dfrac{n_{+,T}\, b}{n_{0,T} N_T} & \text{si } \mathrm{n}_T(c_1c_2) = 0 \end{cases} \tag{3.19}$$

donde $n_{+,T}$ y $n_{0,T}$ se definen formalmente como:

$$n_{+,T} = \sum_{c_1c_2 : \mathrm{n}_T(c_1c_2) \geq 1} 1 \quad \text{y} \quad n_{0,T} = \sum_{c_1c_2 : \mathrm{n}_T(c_1c_2) = 0} 1. \tag{3.20}$$

Es decir, los $n_{+,T}$ pares de clases que han sido vistos al menos una vez en la parte retenida del corpus proporcionan una masa de probabilidad $n_{+,T}\frac{b}{N_T}$ que se distribuye de forma uniforme entre los $n_{0,T}$ eventos no vistos en la parte retenida.

Eliminando un evento c_1c_2 del corpus, se reducirá en uno el número de ocurrencias de dicho evento en todo el corpus. Por lo tanto, si eliminamos un evento que ocurre exactamente una vez, se incrementará el número n_0 de bigramas no vistos en todo el corpus y se decrementará el número n_+ de bigramas visto alguna vez en todo el corpus.

Es posible expresar $\mathrm{P}_{G,T}(c_1c_2)$ en función de las cuentas $\mathrm{n}(\cdot)$, n_0 y n_+ sobre el corpus de entrenamiento completo, análogas a $\mathrm{n}_T(\cdot)$, $n_{0,T}$ y $n_{+,T}$ sobre la parte retenida, como:

$$\mathrm{P}_{G,T}(c_1c_2) = \begin{cases} \dfrac{\mathrm{n}(c_1c_2) - 1 - b}{N - 2} & \text{si } \mathrm{n}(c_1c_2) > 1 \\ \dfrac{(n_+ - 1)b}{(n_0 + 1)(N - 2)} & \text{si } \mathrm{n}(c_1c_2) = 1 \end{cases} \tag{3.21}$$

Por otro lado, para la estimación de $\mathrm{P}_{G,T_c}(c)$, se pueden utilizar las cuentas relativas siempre que se garantice que cada clase ocurre al menos una vez en la parte retenida del corpus. Dicha estimación sería:

$$\mathrm{P}_{G,T_c}(c) = \frac{\mathrm{n}_T(c)}{N_T} = \frac{\mathrm{n}(c) - 1}{N - 2}. \tag{3.22}$$

Una forma de garantizar que cada clase ocurre al menos una vez en el corpus retenido es evitando explícitamente que se pueda mover una determinada palabra w_i de su clase c_i si la nueva $\mathrm{n}(c_i) \leq 1$. Otra forma de conseguirlo es excluir del proceso de agrupamiento aquellas palabras que ocurren un número pequeño de veces por considerarse poco fiables; además, eliminar las palabras menos frecuentes constituye un buen estimador de las palabras no vistas [KN93]. En nuestro caso, debido a que el proceso de agrupamiento se aplica sobre dominios restringidos, hemos preferido tener una mayor cobertura del lenguaje e

incorporar todas las palabras al proceso de agrupamiento garantizando de forma explicita que siempre se cumpla que $\mathrm{n}(c_i) > 1$.

Sustituyendo (3.21) y (3.22) en (3.18), la información mutua entre clases adyacentes cuando se aplica el método de dejar uno fuera, $\mathrm{I}_{lo}(c_1; c_2)$, queda:

$$\begin{aligned}\mathrm{I}_{lo}(c_1; c_2) \simeq \frac{1}{N-1} \Bigg[& \sum_{c_1 c_2 : \mathrm{n}(c_1 c_2) > 1} \mathrm{n}(c_1 c_2) \log \frac{\mathrm{n}(c_1 c_2) - 1 - b}{N-2} \\ & + n_1 \log \frac{(n_+ - 1) b}{(n_0 + 1)(N-2)} \\ & - 2 \sum_c \mathrm{n}(c) \log \frac{\mathrm{n}(c) - 1}{N-2} \Bigg] . \end{aligned} \tag{3.23}$$

El valor del parámetro b se puede estimar como [YB97, pág. 189]:

$$b = \frac{n_1}{n_1 + 2 n_2}. \tag{3.24}$$

Aunque, a efectos prácticos, en lugar de entrenar b para cada agrupamiento, se suele determinar empíricamente un valor apropiado.

Hemos llamado al segundo de los algoritmos propuestos, algoritmo *iterativo dejando uno fuera* de agrupamiento monolingüe, precisamente por que hace uso del método de *dejar uno fuera* para calcular la función objetivo que guía el agrupamiento. De hecho, salvo por la nueva función objetivo, $\mathrm{I}_{lo}(c_1; c_2)$ frente a $\mathrm{I}(c_1; c_2)$, el algoritmo iterativo dejando uno fuera, es idéntico al algoritmo 3.1 descrito en la sección anterior. Así mismo, también utiliza las técnicas descritas en la sección anterior para la reducción del coste temporal puesto que el incremento de $\mathrm{I}_{lo}(c_1; c_2)$ también puede calcularse de forma parcial. La única complejidad añadida es la de tener que calcular la variación de los valores n_+, n_1 y n_0 con cada movimiento evaluado.

A modo de ejemplo de la utilización de este algoritmo, la figura 3.3 muestra las distintas perplejidades obtenidas sobre la parte castellana del corpus castellano–inglés EuTrans I (ver capítulo 6). En la figura se representan: la perplejidad del conjunto de test utilizando un modelo de bigramas obtenido sobre el corpus de entrenamiento; la evolución de la perplejidad del test utilizando modelos de bigramas basados en un número distinto de clases; la perplejidad del entrenamiento utilizando modelos de bigramas basados en un número distinto de clases; y la perplejidad del entrenamiento utilizando un modelo de bigramas. Si se observa la perplejidad medida sobre el test de los modelos de bigramas basados en clases, se puede apreciar como disminuye al principio conforme el número de clases aumenta para después comenzar a ascender. También se puede observar como la perplejidad medida sobre el conjunto de entrenamiento utilizando la técnica de dejar uno fuera presenta un comportamiento similar a la medida sobre el test. Además, si se compara la figura 3.3 con la la figura 3.2

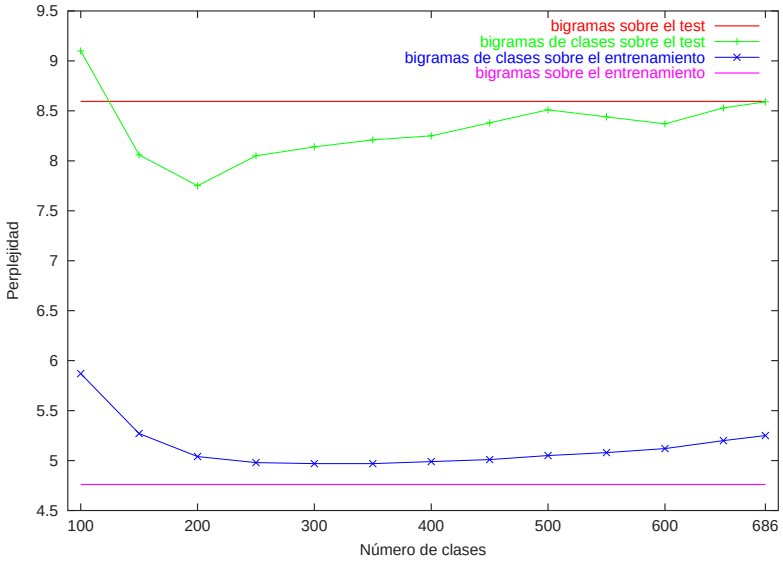

Figura 3.3: Perplejidad medida sobre el test y el entrenamiento para bigramas y bigramas de clases utilizando el algoritmo iterativo dejando uno fuera (comparar con la figura 3.2).

se puede ver como la perplejidad medida sobre el test es generalmente menor cuando se utiliza la técnica de dejar uno fuera. Por último, la perplejidad sobre el test alcanza su mínimo para 200 clases, valor relativamente cercano a las 300 clases en las que alcanza el mínimo la perplejidad sobre el corpus de entrenamiento. Por lo tanto, la perplejidad obtenida utilizando la técnica de dejar uno fuera permite obtener una estimación apropiada del número óptimo de clases.

3.6. Algoritmo incremental

El último de los algoritmos presentados, al que hemos bautizado con el nombre de algoritmo incremental (ver algoritmo 3.2), genera el agrupamiento de forma incremental; comienza con un número reducido de clases y crea nuevas clases, una a una, hasta alcanzar el número deseado de éstas.

Los dos algoritmos anteriores presentan una gran dependencia con la distribución inicial de palabras en clases. Esto es así ya que realizan una serie de movimientos de palabras, a partir de dicha distribución, hasta que se alcanza un máximo de la función objetivo; que puede ser un máximo local. Cuanto mejor sea la distribución inicial, más probabilidades habrá de que el agrupamiento alcanzado sea óptimo. A falta de más información, los dos algoritmos anteriores

optan por la utilización de una distribución inicial que permite que las palabras más frecuentes sean movidas en primer lugar. Asumimos que podemos sacar mejores conclusiones sobre el modelo del lenguaje a partir de aquellas palabras que hemos visto más veces.

Para contrarrestar la dependencia de estos algoritmos con la distribución inicial podríamos generar un número determinado de distribuciones iniciales al azar y realizar agrupamientos para cada una de estas distribuciones. De éstos, consideraríamos como el mejor agrupamiento a aquel que maximizara la función objetivo. Cuanto más grande sea el número de distribuciones iniciales, más probable será que encontremos un mejor agrupamiento. Sin embargo, como se ha visto en (3.2), el número de particiones posibles es extremadamente grande, por lo que esta aproximación no parece demasiado prometedora.

Frente a esta alternativa, planteamos un algoritmo que comienza con un número de clases menor que el que se pretende alcanzar y encuentra el agrupamiento (sub)óptimo para este número inicial de clases utilizando cualquiera de los dos algoritmos anteriores. Partiendo de dicho agrupamiento se crea una nueva clase a la que se mueven aquellas palabras que con su movimiento mejoren la función objetivo. De nuevo, se encuentra el agrupamiento (sub)óptimo para el nuevo número de clases utilizando cualquiera de los algoritmos anteriores. Así, hasta alcanzar el número de clases deseado. De esta forma, cada uno de los agrupamientos realizados comienza con una distribución inicial basada en el agrupamiento anterior y además, puede variar dicha distribución por completo si fuera conveniente.

Podríamos clasificar a este algoritmo como un algoritmo de agrupamiento jerárquico divisivo. Comienza con un determinado número de clases menor al deseado (en el extremo con sólo una clase) y termina cuando alcanza el número deseado de clases o cuando la creación de una nueva clase no mejora la función objetivo utilizada. Al igual que en el algoritmo jerárquico aglomerativo de Brown et al. [BDd$^+$92], y a diferencia de los agrupamientos jerárquicos clásicos, no estamos interesados en la obtención de una jerarquía de clases, únicamente en la distribución final de palabras en clases. Esta es la razón por la que podemos mover palabras de una clase a otra con cada incremento del número de clases.

El objetivo del algoritmo con cada incremento en el número de clases es encontrar una distribución con $n + 1$ clases que mejore la distribución actual de n clases. Para ello, se añade una nueva clase c_{n+1} y se busca qué palabra w_i mejora la función objetivo al ser movida de su clase actual c_i a la nueva clase c_{n+1}. Una vez se ha movido la palabra w_i, se intentan mover las restantes palabras w_j de c_i a la clase c_{n+1}. De esta forma, se reparten los miembros de la clase c_i entre las clases c_i y c_{n+1}. Por último, sobre la nueva distribución de palabras en $n + 1$ clases se realizan tantas iteraciones de cualquiera de los algoritmos descritos en las secciones anteriores como sean necesarias. Cuando no se realizan más movimientos o se alcanza el número de iteraciones prefijado, se incrementa en uno el número de clases y se repite todo el proceso.

Si se utiliza $\mathrm{I}(c_1; c_2)$ como función objetivo, en la forma descrita en la sec-

ción 3.4, tenemos el problema de que dicha función aumenta conforme aumenta el número de clases por lo que el número de clases deseado deberá ser fijado de antemano. Por otro lado, si se utiliza $\mathrm{I}_{lo}(c_1; c_2)$ como función objetivo, en la forma descrita en la sección 3.5, el algoritmo incremental puede parar de forma automática en el momento en el que $\mathrm{I}_{lo}(c_1; c_2)$ no mejore. En la experimentación que hemos realizado, hemos utilizado siempre este método conjuntamente con la función $\mathrm{I}_{lo}(c_1; c_2)$.

Algoritmo 3.2 Agrupamiento incremental

Requiere: C, $maxIter$
  Inicializar $\mathcal{G} : \mathcal{G}(w_i) \leftarrow 1,\ \forall w_i;\ n \leftarrow 1$
  repetir
    Crear una nueva clase c_{n+1} vacía
    Mover tentativamente cada palabra w a la nueva clase c_{n+1}
    Mover la palabra w_i que más incremente $\mathrm{I}_{lo}(c_1; c_2)$ a c_{n+1}
    si existe tal palabra w_i **entonces**
      para cada w_j en la anterior clase de w_i, c_i, **hacer**
        Mover w_j a c_{n+1} si con ello se incrementa $\mathrm{I}_{lo}(c_1; c_2)$
      fin para
      $n \leftarrow n + 1$
      Hacer *iterativo dejando uno fuera*$(n, maxIter)$
    fin
  hasta $n = C$ o $w_i = \emptyset$

3.7. Conclusiones

En este capítulo hemos presentado tres algoritmos de agrupamiento monolingüe. Los dos primeros de ellos difieren en la forma en la que calculan la función objetivo que utilizan. El algoritmo *iterativo* utiliza la información mutua entre clases adyacentes. El algoritmo *iterativo dejando uno fuera* utiliza dicha función objetivo calculada utilizando la técnica de dejar uno fuera para simular eventos no vistos y evitar el sobreentrenamiento. Se espera que el agrupamiento generado por el segundo de los métodos sea mejor que el generado por el primero. Además, también se espera que la forma en la que se calcula la función objetivo en el algoritmo *iterativo dejando uno fuera* permita estimar el número óptimo de clases a partir del corpus de entrenamiento.

El último de los algoritmos, el algoritmo *incremental* puede verse como una extensión de los anteriores que intenta minimizar el efecto de la distribución inicial sobre el agrupamiento obtenido. Comienza con un número de clases menor que el deseado e incrementa el número de clases de una en una hasta alcanzarlo. Se espera que los agrupamientos generados mediante el algoritmo *incremental* sean mejores que los obtenidos con cualquiera de los dos anteriores. Además, se

confía en que los agrupamientos generados por el algoritmo *incremental* para distintos números de clases sean más homogéneos que los obtenidos de forma independiente utilizando cualquiera de los otros dos métodos.