CONSTRUCCIÓN NO LINEAL DE N-GRAMAS EN LA LINGÜÍSTICA COMPUTACIONAL

N-GRAMAS SINTÁCTICOS, FILTRADOS Y GENERALIZADOS

Grigori Sidorov

México 2013

Aprobado con base en proceso de revisión por pares por el Consejo Editorial Académico de la Sociedad Mexicana de Inteligencia Artificial. Revisado y recomendado por el Grupo de Ingeniería Lingüística del Instituto de Ingeniería de la UNAM (jefe del grupo Dr. Gerardo Sierra), grupo IULATERM (léxico, terminología, discurso especializado e ingeniería lingüística) del Institut Universitari de Lingüística Aplicada de la Universitat Pompeu Fabra, Barcelona, España (directora del grupo Dra. M. Teresa Cabré) y Laboratorio "Natural Language Engineering" del Centro de investigación "Pattern Recognition and Human Language Technology" (PRHLT), Universitat Politècnica de València, España (jefe del Laboratorio Dr. Paolo Rosso).

PC
2013

Construcción no lineal de n-gramas en la lingüística computacional / Sociedad Mexicana de Inteligencia Artificial; Grigori Sidorov. – México: Sociedad Mexicana de Inteligencia Artificial, 2013.
166 p.

ISBN 978-607-95367-9-4

1. Español. I. Sociedad Mexicana de Inteligencia Artificial. II. Grigori Sidorov.

Primera edición, 2013

ISBN 978-607-95367-9-4

Impreso y hecho en México / *Printed and made in Mexico*

Prefacio

Este libro es sobre un nuevo enfoque en la lingüística computacional relacionado con la idea de construir los n-gramas de manera no lineal, siendo el enfoque tradicional usar los datos de la estructura superficial de textos —es decir, en la estructura lineal.

En el libro proponemos y sistematizamos la idea del concepto de n-gramas sintácticos que permite utilizar la información sintáctica dentro de los métodos de procesamiento automático tales como clasificación o agrupamiento —es un ejemplo muy interesante de aplicación de la información lingüística en los métodos automáticos (computacionales). A grandes rasgos, la idea es seguir el árbol sintáctico y construir n-gramas basándose en las rutas en este árbol. Existen varios tipos de n-gramas no lineales; los trabajos futuros deberán determinar qué tipos de n-gramas son más útiles y para qué tareas del PLN.

El libro está dirigido antes que nada a los especialistas en el campo de la lingüística computacional. Sin embargo, hemos hecho un esfuerzo para explicar de manera clara cómo utilizar los n-gramas, estamos dando muchos ejemplos, por lo

que creemos que el libro también sirve para los alumnos de posgrado que ya tienen algunos conocimientos previos en el campo. Queremos enfatizar que no se requieren conocimientos profundos de la computación o de las matemáticas; los conceptos que proponemos son intuitivamente muy claros y utilizamos muy pocas fórmulas —bien explicadas, en su caso—.

Grigori Sidorov
Septiembre 2013

Índice general

Índice detallado

Introducción

En este libro estamos discutiendo una idea muy poco estudiada dentro del campo de la lingüística computacional: la construcción de los n-gramas de manera no lineal.

Antes que nada discutimos a nivel detalle el concepto del modelo de espacio vectorial —un marco conceptual para comparación de cualquier tipo de objetos— y después su aplicación a las tareas relacionadas con textos, es decir, su uso en la lingüística computacional. Se discuten los conceptos relacionados con frecuencia de palabras (*tf-idf*) y se presenta brevemente el análisis semántico latente que permite reducir el número de dimensiones.

Mencionamos los conceptos importantes para el diseño de experimentos en la lingüística computacional y describimos el esquema típico de experimento en esta área.

Presentamos el concepto de n-gramas tradicionales (lineales) y lo comparamos con el

concepto de n-gramas obtenidos de manera no lineal: n-gramas sintácticos, filtrados y generalizados.

Los n-gramas sintácticos son n-gramas que se construyen siguiendo las rutas en un árbol sintáctico. Su gran ventaja consiste en que permiten introducir información puramente lingüística (sintáctica) en los métodos computacionales de aprendizaje automático. Su desventaja está relacionada con la necesidad de realizar el análisis sintáctico automático previo.

Consideramos los n-gramas sintácticos tanto continuos como no continuos. En caso de los n-gramas sintácticos continuos, durante su construcción no se permiten bifurcaciones en las rutas sintácticas. Al quitar esta limitación se obtienen los n-gramas sintácticos no continuos: se consideran todos los sub-árboles de longitud *n* de un árbol sintáctico. Cabe mencionar que los n-gramas sintácticos continuos es un caso particular de los n-gramas sintácticos no continuos.

Se propone un metalenguaje para la representación de los n-gramas sintácticos no continuos, es decir, la manera formal de escribir un n-grama sintáctico no continuo usando corchetes y comas, por ejemplo, “a b [c [d, e], f]”. En este caso los corchetes y las comas forman parte de los n-gramas.

También presentamos en este libro varios ejemplos de construcción de n-gramas sintácticos continuos y no continuos para los árboles sintácticos obtenidos usando el sistema FreeLing y el analizador sintáctico de Stanford, tanto para el español como para el inglés.

Estamos mostrando que la aplicación de los n-gramas sintácticos en una de las tareas tradicionales de la lingüística computacional —la tarea de atribución de autoría— da mejores resultados que el uso de los n-gramas tradicionales.

Al final se presentan ideas de otro tipo de construcción no lineal de n-gramas: 1) n-gramas filtrados: se hace un filtrado de palabras o caracteres usando algún criterio antes de construir n-gramas; ésos se construyen usando solamente los elementos que pasaron el filtrado; 2) n-gramas generalizados: las palabras "se generalizan" usando relaciones léxicas, especialmente la sinonimia y la hiperonimia, de esa manera se reduce el universo de elementos que se usan para la construcción de n-gramas.

Se necesitan múltiples estudios experimentales para determinar qué parámetros de construcción de los n-gramas sintácticos continuos y no continuos, filtrados y generalizados son mejores y para qué tipo

de tareas existentes dentro de la lingüística computacional.

El libro sistematiza las ideas recién propuestas por el autor sobre la construcción no lineal de n-gramas para su uso en el modelo de espacio vectorial; por lo que algunas partes del libro se basan en los trabajos previos del autor publicados en varias revistas y congresos, con las actualizaciones y adecuaciones necesarias.

Trabajo realizado con el apoyo parcial del gobierno de México (CONACYT, SNI), de gobierno de la Ciudad de México (proyecto ICYT-DF PICCO10-120), Instituto Politécnico Nacional, México (proyectos SIP 20120418, 20131441, COFAA), y proyecto "Web information quality evaluation initiative" de la Comisión Europea (FP7-PEOPLE-2010-IRSES European Commission project 269180).

PARTE I.
MODELO DE ESPACIO VECTORIAL EN EL ANÁLISIS DE SIMILITUD ENTRE TEXTOS

Capítulo 1.
Formalización en la lingüística computacional

La lingüística computacional

La lingüística computacional es un área importante dentro del campo de la lingüística. Los métodos computacionales que se usan en la lingüística computacional provienen de la ciencia de la computación, o siendo más específico de la inteligencia artificial. Sin embargo, el objeto primordial del estudio de la lingüística computacional sigue siendo la modelación del lenguaje humano, y por lo mismo se sigue situando en el área de las humanidades.

La lingüística computacional estudia cómo construir los modelos del lenguaje de tal manera que sean entendibles para las computadoras, eso quiere decir que no solo se analiza el uso del lenguaje en el comportamiento humano, sino también se aplican

unos métodos formales que permitan la formulación exacta de las hipótesis y su posterior verificación automática utilizando los datos lingüísticos —los corpus [16, 28, 35].

La lingüística computacional y la inteligencia artificial

Precisamente la parte formal de la lingüística computacional se relaciona con los métodos de la inteligencia artificial [27]. De manera muy general se puede definir el propósito de la ciencia de la inteligencia artificial como modelar formalmente la inteligencia humana. Es decir, las preguntas que responde la inteligencia artificial son: ¿qué es el comportamiento inteligente? ¿Cómo los humanos resuelven diariamente miles de problemas prácticos, en la gran mayoría de los casos sin equivocarse? No todas las áreas de inteligencia artificial están relacionadas con el lenguaje humano, por ejemplo, la visión o la resolución de problemas, etc. Sin embargo, en la inteligencia artificial se han desarrollado varios métodos que son aplicables a cualquier tipo de datos: son los métodos de aprendizaje automático; y precisamente esos métodos se aplican en la lingüística

computacional moderna, convirtiéndola en una ciencia formalizada. Hasta cierto grado la lingüística computacional se convierte en una ciencia empírica, donde las hipótesis se verifican con base en los experimentos.

Formalización en la lingüística computacional

Las computadoras son las herramientas modernas más importantes jamás creadas por la humanidad. Sin embargo la naturaleza de las computadoras consiste en una lógica binaria muy sencilla —ceros y unos, más las operaciones lógicas sobre ellos. ¿Cómo traducir un fenómeno tan complejo como un lenguaje humano a esa lógica tan simple? Para eso la lingüística computacional aprovecha tanto los conocimientos que tenemos sobre el lenguaje humano, como las herramientas que existen en el área de las ciencias de la computación y en las matemáticas: varios tipos de modelos formales, lenguajes de programación, etc. [6]

De hecho es curioso, que muchos estudios modernos en el campo de la lingüística computacional, se parecen cada vez más a otras áreas

de la ciencia de la computación y la inteligencia artificial, especialmente a las áreas relacionadas con el aprendizaje automático: clasificación automática o agrupamiento automático. Sin embargo, insistimos, sin la parte lingüística esos métodos no pueden ser aplicados en los modelos de lenguaje humano. En este caso la parte relacionada con la lingüística consiste en cómo elegir las características (y sus valores) que se presentan a los algoritmos de clasificación y agrupamiento. Entonces ¿por qué se dio este paso hacia el uso de los métodos formales justamente en los últimos años? Desde nuestro punto de vista eso está relacionado con los avances de Internet donde en la actualidad existe un gran número de textos libremente disponibles. Esos textos constituyen una excelente fuente de aprendizaje para los sistemas automáticos. Aunque pareciera ser que los sistemas automáticos modernos pueden hacer magia —toman los datos, y realizan las tareas de manera muy similar a un ser humano—, en realidad, se basan en la aplicación de los métodos de aprendizaje automático a unos conjuntos de datos muy grandes.

Para poder aplicar los métodos de aprendizaje automático, la manera más utilizada —y parece ser la única en la actualidad— es aplicar el modelo de

espacio vectorial. Obvio que es uno de los modelos más utilizados en la lingüística computacional moderna. En los siguientes capítulos explicamos brevemente este concepto y discutimos sobre los posibles valores que pueden ser representados en este espacio vectorial para el caso de textos.

Capítulo 2.
Modelo de espacio vectorial

La idea principal del modelo de espacio vectorial

Antes que nada, todo lo que vamos a explicar a continuación es muy sencillo y no requiere los conocimientos de la computación, más bien solamente algo de sentido común. Entonces, sugerimos al lector continuar sin miedo a encontrarse con las matemáticas difíciles, trataremos de explicar las pocas fórmulas que aparezcan lo más claramente posible.

El modelo de espacio vectorial es un modelo utilizado muy ampliamente en las ciencias de la computación. Su amplio uso se debe a la simplicidad del modelo y de su muy clara base conceptual que corresponde a la intuición humana en el procesamiento de información y de datos. Realmente

la idea detrás del modelo es muy sencilla y es una respuesta a la pregunta ¿cómo podemos comparar los objetos de manera formal? Parece que la única manera de describir los objetos es utilizar la representación con los rasgos (características) y sus valores. Es una idea universal, y hasta parece que es la única manera posible de trabajar con los objetos de manera formal [49].

Quizá otra posibilidad sería el uso de las memorias asociativas [27], en cuyo caso en lugar de las características y sus valores se usan las relaciones entre objetos —expresadas a través de las neuronas artificiales con sus respectivas conexiones— y sus pesos. Sobre este tema existe literatura abundante, y no vamos a discutirlo en este libro.

Bueno, ahora ya sabemos representar dos objetos (o más) eligiendo las características que ellos tienen y los valores de esas características. Nótese que en este caso elegir las características equivale a construir un modelo de nuestros objetos. En este sentido la selección de las características es subjetiva, aunque la posterior comparación ya es completamente objetiva.

También es necesario mencionar que la selección de la escala de valores de las características también afecta al modelo que estamos construyendo; y es una

decisión que debemos tomar, por ejemplo, no es lo mismo pesar en gramos o en toneladas.

Un ejemplo del modelo de espacio vectorial

Para tener la idea completamente clara, consideraremos un ejemplo. Supongamos que queremos comparar dos libros. ¿Qué características podemos elegir? Como ya hemos mencionado, eso en gran medida depende de nuestras necesidades, es decir, no existe un conjunto único y correcto de las características de los objetos; sin embargo, hay algunas características más comunes que otras, por ejemplo, en caso de los libros, el número de páginas sería una característica importante para muchos propósitos. También podrían ser las características como color de la portada, autor, editorial, el perfil sociológico de las personas a quienes gustó este libro, etc.

En el caso del número de páginas, el valor de esta característica sería numérico, es decir, se representa por un número. En caso de la editorial, sería una lista de las posibles editoriales. En caso de los perfiles sociológicos, los valores de esa

característica serían un poco más difíciles de representar, por ejemplo, “estudiantes de licenciatura del primer año”, o “profesores universitarios de 40 a 50 años”, etc. Otra vez queremos subrayar que la selección tanto de características como de sus valores queda a nuestra elección, es decir, nosotros estamos construyendo el modelo y sería cuestión de su aplicación práctica ver si es útil o no. Esta idea está muy clara para las características, pero también en muchos casos se aplica a los valores. Por ejemplo, podemos medir el peso en kilogramos o en gramos —esa decisión afectará en gran medida las posteriores comparaciones entre objetos.

Ahora bien, tenemos la representación de dos objetos en términos de las características y de sus valores. ¿Cuál es el paso siguiente? ¿Qué más se necesita para construir un modelo de espacio vectorial? ¿Debe ser algo muy complejo? Realmente no lo es, de hecho el modelo ya está construido. Es un espacio de N dimensiones, y cada dimensión en este espacio corresponde a una de las características: el número de dimensiones es igual al número de las características que tiene el objeto en nuestro modelo.

Se puede imaginar una dimensión como un eje, en el cual podemos marcar los valores de esa

característica/dimensión. Si los valores son numéricos la interpretación es muy clara —es la distancia desde el punto con coordenadas (0,0,...). Si los valores son naturalmente ordenados, como, por ejemplo, los intervalos de la edad, también está claro cómo tratarlos, aunque tenemos que asignarles algunos valores numéricos. Si los valores no son relacionados, como por ejemplo el color de la portada de un libro, la solución más fácil es utilizar un orden aleatorio, e igualmente asignar los valores numéricos a cada valor simbólico.

Nótese que si queremos manejar correctamente esta situación, podemos introducir tantas nuevas dimensiones (características) cuantos valores de la característica tenemos, y sus valores pueden ser "presente-ausente (1-0)". La ventaja es que en este caso no tenemos que ordenar los valores, la desventaja es que se aumenta mucho el número de dimensiones.

El siguiente paso está relacionado con la pregunta: ¿y dónde están los vectores, por qué es un espacio vectorial? Como ya hemos mencionado, cada objeto es un conjunto de los valores de sus características, lo que corresponde a exactamente un punto en un espacio de *N* dimensiones (las

dimensiones corresponden a las características). Este punto exactamente corresponde a un vector (una flecha en la representación geométrica) de *N* dimensiones (un vector n-dimensional), que empieza en el punto con coordenadas (0,0,0,..) en este mismo espacio. Por ejemplo, consideremos un libro de 100 páginas y con color de la portada *rojo*, comparándolo con otro libro de 50 páginas y con el color de la portada *verde*. Este es un espacio de dos dimensiones: "número de páginas" y "color de la portada", y cada libro es un punto con las coordenadas [100, *rojo*] y [50, *verde*]. Nótese que tenemos que elegir, qué valores numéricos corresponden a *rojo* y *verde*.

Las dimensiones corresponden a la posición del valor correspondiente en el vector, es decir, la característica número 1 tiene la posición 1 en el vector, la característica número 2 tiene la posición 2, etc. En este caso como todas las dimensiones son iguales, el ordenamiento de las características nunca afecta el modelo.

Otra pregunta es ¿cómo podemos representar formalmente los espacios vectoriales? Como ya mencionamos, cada objeto es un conjunto de sus valores de las características seleccionadas; por ejemplo, uno de los libros se presenta como

X=[(*número de páginas=100*), (*color de portada=rojo*)], y el otro Y=[(*número de páginas=50*), (*color de portada=verde*)]. Surgen dos preguntas: 1) ¿hay que repetir cada vez el nombre de la característica? y 2) ¿qué se puede hacer en caso que algún objeto no tenga alguna característica?

Cuadro 1. Ejemplo de representación tabular del espacio vectorial.

	Libro *X*	**Libro *Y***
Dimensión 1: ***Número de páginas***	*100*	*50*
Dimensión 2: ***Color de portada***	*rojo*	*verde*

La respuesta de la primera pregunta es muy sencilla, normalmente se omite el nombre de la característica: de esta manera X=[*100*, *rojo*] y Y=[*50*, *verde*]. Recordamos que el valor de la misma característica tiene el mismo índice en el vector, es decir, la misma posición en el vector. Otra explicación puede basarse en un cuadro, donde las columnas corresponden a los objetos y las filas a las características (o puede ser al revés, eso no cambia el modelo), véase el Cuadro 1. En este sentido, representación tabular (una matriz) y representación

vectorial es lo mismo. Las columnas corresponden a los objetos, y las filas a las características, mientras el valor de cada celda es el valor de la característica de un objeto dado.

En este sentido, el mero hecho de saber en qué columna se encuentra un valor ya define a qué característica este valor corresponde; es decir, esa información se sabe simplemente por su posición en un cuadro.

Igualmente sencilla es la respuesta para la segunda pregunta, qué hacer en caso de que un objeto simplemente no tenga definida una característica dada: se ponen los valores iguales a cero en el lugar correspondiente. En todos los cálculos posteriores esos valores no afectarán el resultado siendo iguales a cero.

Nótese que la representación tabular es conceptualmente lo mismo que el modelo de espacio vectorial *per se*: las columnas corresponden a las dimensiones. La única diferencia es que en este caso no es tan natural el uso de los conceptos geométricos que vamos a presentar a continuación, pero la ventaja de esa representación consiste que en el futuro podemos aplicar los métodos de álgebra lineal —por ejemplo, el análisis semántico latente (*LSA*)—. Vamos

a considerar un ejemplo de aplicación de esos métodos más adelante.

Similitud de objetos en el modelo de espacio vectorial

Ahora ¿qué ventajas nos da el concepto de espacio vectorial? Ya lo tenemos, ¿en qué nos ayuda? Resulta que podemos utilizar la metáfora de espacio para calcular la similitud entre objetos, es decir comparar los objetos, basándonos en puras ideas geométricas muy sencillas, no más allá del teorema de Pitágoras, como explicaremos más adelante.

Ahora bien, cada objeto es un vector en un espacio de *N* dimensiones. ¿Cómo comparar esos vectores? La idea geométrica es que los vectores que tienen más o menos la misma dirección se parecen entre sí. Hablando de manera más formal, si menor es el ángulo entre esos vectores, mayor es la similitud. Es muy claro e intuitivo en un espacio de dos dimensiones. Por ejemplo, en la Ilustración 1, menor es el ángulo entre cada par de flechas, más "se parecen entre sí" ambas flechas de este par, es decir, más coincide su dirección.

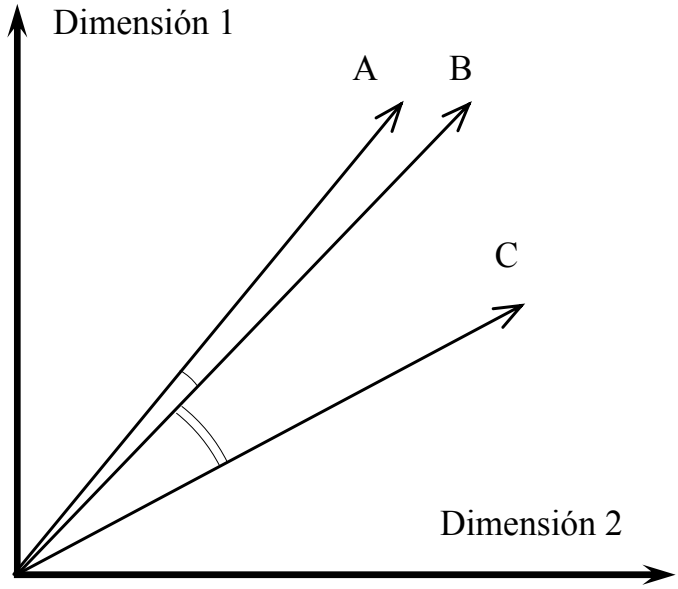


Ilustración 1. Ejemplo del espacio vectorial: similitud entre vectores.

Por ejemplo, los vectores A y B se parecen entre sí más que los vectores B y C. Y por supuesto se parecen menos entre sí los vectores A y C. Nótese que en total se puede comparar tres pares de vectores, en caso de tener tres vectores.

En un espacio con mayor número de dimensiones es más difícil imaginar esa similitud, por eso sugerimos al lector siempre considerar los ejemplos en un espacio de dos dimensiones; tomando en cuenta que en un espacio con mayor número de dimensiones las ideas serían exactamente iguales.

Similitud de coseno entre vectores

Para expresar formalmente la similitud, se utiliza la medida del coseno del ángulo entre los vectores: menor el ángulo, mayor es su coseno; es decir, mayor es la similitud entre vectores —y de esa manera de los objetos mismos que estamos comparando—.

Para calcular la similitud de coseno entre dos vectores, llamémoslos V y U, se utiliza el producto interno (producto punto) de los vectores normalizados. La **normalización** consiste en la división del resultado entre la longitud de cada vector o, qué es lo mismo, de la multiplicación de sus longitudes. La longitud se llama la **norma euclidiana** y se denomina, por ejemplo, para el vector V como $\|V\|$.

Nos gustaría recordar al lector qué significa "normalizar" y por qué es importante. A menudo es importante no comparar los valores absolutos, sino los valores relativos. Por ejemplo, qué es mayor, ¿10 o 20? La pregunta pareciera no tener sentido, porque obviamente 20 es mayor que 10. Sin embargo ¿si supiéramos que ese 10 está en la escala de 30, y 20 está en la escala de 100? En este caso, si normalizamos esos valores: 10/30 es mayor que

20/100. Eso demuestra que si al comparar valores se toma en cuenta la escala y se busca la misma escala para ambos, los resultados dependen de la normalización (escala). En caso de los vectores en dos dimensiones, un vector puede ser una flecha muy larga y el otro una flecha muy corta. Para poder comparar su similitud, debemos convertirlos en los vectores unitarios aplicando la norma euclidiana.

El producto punto de ambos vectores es un valor que es muy sencillo de obtener: se suman las multiplicaciones de los valores de vectores en cada dimensión. En dos dimensiones —dimensión 1 y dimensión 2— para los vectores V y U, eso sería:

$$ProductoPunto = V_1 \times U_1 + V_2 \times U_2.$$

Es decir, se multiplican entre sí los primeros elementos de los vectores, después se multiplican entre sí los segundos elementos de los vectores, y se suman al final.

De manera más general:

$$ProductoPunto\ (V, U) = \sum_{n=1}^{m} (V_n \times U_n),$$

donde m es el número de dimensiones en el modelo de espacio vectorial (que es igual al tamaño de vectores).

Como ya mencionamos, el segundo paso en el cálculo de coseno después de calcular el producto punto es normalizar los vectores. Para eso se calcula su longitud y se obtiene la norma euclidiana. La norma euclidiana convierte cada vector a un vector de longitud uno. La normalización utilizando la norma euclidiana $\|V\|$ consiste en la división del vector entre su longitud. La longitud del vector se calcula como

$$\|V\| = \sqrt{\sum_{k=1}^{m} V_k^2}.$$

En caso de dos dimensiones, eso corresponde a claramente a la aplicación del teorema de Pitágoras:

$$\|V\| = \sqrt{{V_1}^2 + {V_2}^2},$$

donde V_1 y V_2 son valores del vector V en el eje 1 y eje 2. Realmente, V_1 es el valor del vector en la dimensión 1, es decir, la longitud del primer cateto; y V_2 es el valor del vector en la dimensión 2, es decir, la longitud del segundo cateto, ver la Ilustración 1. En este caso, el vector como tal corresponde a la hipotenusa, y se aplica el teorema de Pitágoras.

En caso de un mayor número de dimensiones, se agregan los elementos correspondientes en la fórmula de la misma manera.

Entonces la fórmula final para el cálculo de similitud de coseno consiste en obtener el producto punto de los dos vectores y después aplicar la norma euclidiana. De manera general:

$$\mathrm{sim}(V,U) = \frac{\sum_{n=1}^{m} (V_n \times U_n)}{\|V\| \times \|U\|}.$$

En este caso, la similitud de coseno nos indica que tan parecidos son los vectores V y U. Para los valores positivos, el coseno está en el rango de 0 a 1.

Note que la similitud de coseno está definida exactamente para dos objetos (dos vectores). La similitud del objeto consigo mismo sería igual a 1. Para objetos a los cuales corresponden los vectores ortogonales (es decir, el ángulo entre ellos es 90°), la similitud de coseno es igual a 0.

Las ideas que presentamos en este capítulo son muy sencillas, sin embargo permiten comparar cualquier tipo de objetos utilizando 1) el modelo de espacio vectorial, 2) la correspondiente metáfora espacial, y 3) las ideas geométricas muy sencillas.

Capítulo 3.
Modelo de espacio vectorial para textos y la medida *tf-idf*

Las características de textos para el modelo de espacio vectorial

Ahora bien, ya sabemos qué es un modelo de espacio vectorial para el caso de cualquier tipo de objetos. Este modelo consiste en selección de las características y asignación de los valores de esas características, que permite representar nuestros objetos como vectores (precisamente de los valores de las características), y después medir su similitud utilizando una simple fórmula de similitud de coseno. Recordamos que para el cálculo de similitud hay que considerar exactamente dos objetos, en caso de tener mayor número de objetos se realizan los cálculos de

similitud (las comparaciones entre objetos) en parejas.

Vamos a ver cómo se aplica este modelo para la comparación de documentos (textos). Es decir, nuestros objetos que queremos comparar son documentos.

La necesidad de medir la similitud es una situación muy típica en tareas del procesamiento automático de lenguaje natural y de la lingüística computacional. Por ejemplo, la tarea más general de recuperación de información consiste precisamente en el cálculo de similitud. Vamos a explicar esto con un poco más de detalle.

En recuperación de información se cuenta con una colección de documentos, posiblemente muy grande. Para el caso de los buscadores en Internet, esa colección consiste en todos los textos de Internet indexados previamente por las "arañas", es decir, los programas que siguen las ligas en Internet (la "telaraña mundial"). El usuario formula una petición, que también tiene una forma textual. En este caso, la tarea de recuperación consiste en encontrar en la colección los documentos lo que "se parezca" más a la petición, es decir, son similares con la petición [4]. A menudo, como criterios adicionales, se utilizan otros

criterios de similitud como los perfiles de usuarios o la estructura de la colección (por ejemplo, como en el algoritmo *PageRank*).

Ahora bien, si queremos comparar los documentos, y ya sabemos que la manera más recomendada es utilizar el modelo de espacio vectorial, ¿cómo elegir las características que tienen los documentos? ¿Qué deben ser esas características y cuáles son sus valores? Igual que siempre sucede con el modelo de espacio vectorial, tenemos muchas opciones y depende de nosotros qué características vamos a considerar importantes y cómo vamos a elegir sus valores. Es decir, en el modelo de espacio vectorial la comparación es objetiva, pero la selección de las características y selección de escalas de sus valores son subjetivas.

La idea más sencilla es utilizar palabras como características de documentos. Normalmente se aplican algunos procedimientos adicionales, como por ejemplo, la lematización, es decir, todas las formas se sustituyen por sus lemas —formas normalizadas, por ejemplo, *trabajaron, trabajamos, trabajen* etc. tienen el lema *trabajar*—. También muy a menudo de cálculo de similitud se excluyen las palabras auxiliares (también llamados en inglés *stop words*), tales como

preposiciones o artículos, porque su presencia en un documento no dice nada del documento como tal, su presencia es determinada por las características del lenguaje. Es usual para muchas tareas, a excepción de tener una tarea específica que requiera la presencia de ese tipo de palabras precisamente como podría ser, por ejemplo, para el análisis de estilo [2].

Valores de las características textuales: *tf-idf*

Si las características son palabras, ¿qué podemos utilizar como sus valores? Intuitivamente, puede ser algo relacionado con frecuencia de palabras. En este sentido, entre más frecuente es la palabra, más importante es esta palabra para el documento. No todo es tan sencillo, pero la idea es ésa.

La frecuencia de una palabra se denomina *tf* (*term frequency*), es decir, frecuencia de la palabra (término) significa cuántas veces la palabra ocurre en un documento. Más específicamente se denomina tf_{ij}, es decir, cuántas veces la palabra i aparece en el documento j. En este sentido, es un valor que puede ser diferente en cada documento de la colección.

Normalmente la frecuencia de una palabra se combina con otra medida, que se llama *idf* (en inglés, *inverse document frequency,* frecuencia inversa de documentos). La intuición detrás del *idf* se relaciona con lo siguiente: si una palabra se encuentra en todos los documentos de nuestra colección, entonces esta palabra es incapaz de distinguir entre los documentos, y por lo tanto no nos sirve. Y al contrario, si una palabra se encuentra exactamente en un documento en nuestra colección, es una palabra muy útil para cálculos de similitud (o, por ejemplo, recuperación de información, que, como ya mencionamos, es un caso particular de cálculo de similitud). El *idf* se calcula para cada palabra en una colección dada, es decir, depende de la colección, pero no depende de un documento específico en la colección.

La fórmula para el cálculo del *idf* es la siguiente:

$$idf_i = \log \frac{N}{DF_i},$$

donde N es el número total de documentos en la colección de documentos, DF_i es el número de documentos donde la palabra i se encuentra por lo menos una vez —es decir, en este caso no importa que

tan frecuente es dentro del documento, con que se encuentre una vez basta; note que es un convenio verificado por la práctica—.

Se puede observar, que si una palabra se encuentra en todos los documentos, entonces el valor de *idf* es igual a $\log(\frac{N}{N})$, que es igual a 0. El valor *idf* es máximo cuando una palabra se encuentra exactamente en un documento.

Se utiliza el logaritmo nada más para suavizar la influencia de las frecuencias altas; es un uso típico de logaritmos en la lingüística computacional. Note que DF_i nunca puede ser cero, porque solo hemos considerado las palabras que están presentes en nuestra colección. También podemos utilizar en lugar del *tf*, su logaritmo: $\log(tf + 1)$, en este caso tenemos que prever la posibilidad de cero, por eso usamos "+1".

En caso general se recomienda combinar *tf* e *idf* de una palabra dada en cada documento: normalmente se multiplican. La medida que combina esas dos características se llama *tf-idf* (se pronuncia "*te-efe-i-de-efe*") y muy a menudo se utiliza como el valor que tienen las palabras como características en el modelo de espacio vectorial para comparar los documentos. También se puede probar con el *tf* solamente, el *tf* normalizado, el *idf*, etc. [28].

Matriz término-documento

Entonces, nuestros objetos que estamos considerando son documentos (textos). Tenemos palabras (términos) como características de esos documentos, cada palabra tiene un valor *tf-idf* en cada documento. Eso quiere decir que a cada documento le corresponde un vector de valores *tf-idf* de las palabras.

También podemos representar toda esa información como una matriz. Esa matriz se llama "matriz término-documento". Se presenta un ejemplo en la Ilustración 2.

	Doc_1	Doc_2	Doc_3	Doc_4...
$Palabra_1$	0.3	0	0	0.02
$Palabra_2$	0.7	0.01	0	0.5
$Palabra_3$	0	0	0	0
$Palabra_4$	0	0	2.2	0
$Palabra_5$	1.2	0	0	0
...				

Ilustración 2. Una matriz término documento.

Los documentos $Doc_1 ... Doc_N$ representan los documentos en nuestra colección, las palabras $Palabra_1 ... Palabra_M$ representan todas las palabras

que se encuentran en esos documentos ordenadas de alguna manera. Los valores en el cuadro corresponden a los supuestos valores *tf-idf* en la colección.

Nótese que típicamente el cuadro es muy disperso, es decir tiene muchos ceros. Se recomienda utilizar el índice invertido para su representación. El índice invertido consiste en conversión del cuadro en una lista. Por ejemplo, la lista invertida para la ilustración anterior sería:

(Doc_1, $Palabra_1$, 0.3), (Doc_1, $Palabra_2$, 0.7), (Doc_1, $Palabra_5$, 1.2), (Doc_2, $Palabra_2$, 0.01), (Doc_3, $Palabra_4$, 2.2), (Doc_4, $Palabra_1$, 0.02), (Doc_4, $Palabra_2$, 0.5),

En este caso no es necesario mantener en la lista ningún elemento para los valores iguales a cero. La matriz inicial y el índice invertido contienen la misma información. Es un procedimiento estándar que siempre debe aplicarse a las matrices dispersas.

Es fácil observar que esta representación es igual a la representación con vectores, por ejemplo, al documento Doc_1 según se representa en la Ilustración 2, le corresponde el vector [0.3, 0.7, 0, 0, 1.2].

Vamos a utilizar la representación matricial (tabular) en la discusión sobre el análisis semántico latente.

N-gramas tradicionales como características en el modelo de espacio vectorial

Nótese que en caso de utilizar palabras como características se pierde la información sobre las relaciones sintácticas entre palabras. Las palabras se convierten en lo que se llama "bolsa de palabras" (en inglés, *bag of words*). Para muchas tareas esa pérdida de información es aceptable. Más adelante en el libro vamos a proponer una posible solución para evitar esa pérdida de información sintáctica —los n-gramas sintácticos.

¿Qué más, a parte de las palabras, podrían ser las características de documentos? Quizá, es difícil inventarlo rápido desde cero, pero el concepto como tal es muy sencillo. Se trata de los n-gramas (tradicionales). La sencillez de los modelos que describimos es algo que ya hemos afirmado sobre el modelo de espacio vectorial, y creemos que estamos convenciendo al lector de que realmente es así.

N-gramas tradicionales son secuencias de elementos tal como aparecen en un documento [35]. En este caso la letra *N* indica cuántos elementos

debemos tomar en cuenta, es decir, la longitud de la secuencia o de n-grama. Por ejemplo, existen bigramas (2-gramas), trigramas (3-gramas), 4-gramas, 5-gramas, etc. De esa manera, si vamos a hablar de unigramas, es decir, de n-gramas construidos de un solo elemento, es lo mismo que hablar de palabras.

Otro grado de libertad es el tipo de elementos que están formando n-gramas. Pueden ser lemas o palabras, pero también pueden ser etiquetas de clases gramaticales (en inglés, *POS tags*, *part of speech tags*), tales como sustantivos, verbos, etc. Posiblemente las etiquetas pueden incluir características gramaticales más detalladas, por ejemplo, una etiqueta como VIP1S, podría significar "verbo, indicativo, presente, primera persona, singular". Podemos construir n-gramas utilizando ese tipo de etiquetas.

En los últimos años en algunas tareas se utilizan mucho los n-gramas de caracteres, es decir secuencias de caracteres tomadas de un texto. Curiosamente, para algunas tareas, como atribución de autoría, los n-gramas de caracteres dan buenos resultados [52]. Su interpretación lingüística no está clara, y sigue siendo el objeto de estudios.

Vamos a ver un ejemplo de los n-gramas tradicionales de palabras. De la frase: *Juan lee un libro interesante* podemos sacar los siguientes bigramas (2-gramas): *Juan lee, lee un, un libro, libro interesante*. O los siguientes trigramas (3-gramas): *Juan lee un, lee un libro, un libro interesante*, etc. Podemos sustituir cada palabra por su lema o por su clase gramatical y construir los n-gramas correspondientes. Como se puede observar, el procedimiento es muy sencillo, pero se usa con gran éxito en los sistemas de lingüística computacional.

Y si utilizamos los n-gramas como características, ¿qué valores ellos pueden tener? Igual que en caso de las palabras (unigramas), son los valores relacionados con su *tf-idf*. Nótese que las frecuencias de n-gramas usualmente son mucho más bajas que de las palabras, es decir los n-gramas aparecen mucho menos en los textos. Es lógico porque realmente estamos observando la aparición al mismo tiempo de las secuencias de dos o más palabras seguidas, lo que es un evento mucho menos probable que de una palabra sola.

Entonces, para aplicar el modelo de espacio vectorial a los textos, podemos utilizar como características los n-gramas de varios tamaños

compuestos de elementos de varios tipos, y como sus valores utilizar sus frecuencias *tf*, *idf* o su combinación *tf-idf*.

Capítulo 4.
Análisis semántico latente (LSA): reducción de dimensiones

La idea del análisis semántico latente

Después de construir el modelo de espacio vectorial se puede representar y comparar cualquier tipo de objetos de nuestro estudio. Ahora podemos discutir la cuestión si podemos mejorar el espacio vectorial que hemos construido de alguna manera. La importancia de esa pregunta está relacionada con el hecho que el modelo de espacio vectorial puede tener miles y miles de características, y posiblemente muchas de esas características son redundantes. ¿Hay alguna manera de deshacernos de las características que no son tan importantes?

Para analizar la dependencia de las características entre sí existen varios métodos, por ejemplo, el análisis de los componentes principales (PCA, *principle component analysis*), el coeficiente de correlación, etc. En este capítulo vamos a describir brevemente el método que se llama análisis semántico latente (LSA, *latent semantic analysis*, también a veces llamado LSI, *latent semantic indexing*) [10].

Antes que nada debemos aclarar que aunque la idea del análisis semántico latente aplicado a los textos es encontrar las palabras que se comportan de manera similar (basándose en el análisis de sus contextos), y en este sentido tienen la misma semántica, en la gran mayoría de los casos los resultados no tienen nada que ver con la semántica, salvo la intención inicial. Es decir, se busca la similitud semántica distribucional, pero es muy difícil en la práctica que el análisis semántico latente realmente la encuentre.

En realidad, el análisis semántico latente es aplicación de una técnica de procesamiento de matrices tomada de álgebra lineal. Esta técnica se llama descomposición en valores singulares (*SVD*, *singular value decomposition*) y lo que permite es encontrar en las matrices aquellas filas con mayor

información (valores grandes) y así eliminar las filas de menor información (valores pequeños).

En este sentido, esencialmente para nuestro caso el análisis semántico latente no es nada más que una técnica de reducción de dimensionalidad de un espacio vectorial. Bueno, ¿y por qué de repente estamos hablando de las matrices? ¿Y cómo las matrices están relacionadas con las dimensiones? Ya hemos discutido este aspecto en la sección relacionada con la matriz término-documento: los objetos se representan como vectores —lo que corresponde a un espacio multidimensional—, pero el conjunto de los vectores representa a una matriz.

Ejemplos de aplicación del análisis semántico latente

No vamos a entrar en este libro en los detalles matemáticos del análisis semántico latente. En lugar de eso vamos a dar un par de sencillos ejemplos. La intuición detrás del análisis semántico latente se puede representar de dos maneras.

Vamos a considerar una matriz que caracteriza cuatro objetos O_1-O_4 y utiliza cuatro características

c_1-c_4. Se puede observar que los valores de cada par de las características se repiten. En este sentido, una característica en cada par es redundante, ver la Ilustración 3.

	O_1	**O_2**	**O_3**	**O_4**
c_1	1	1	0	0
c_2	1	1	0	0
c_3	0	0	1	1
c_4	0	0	1	1

$\Rightarrow$ LSA

	O_1	**O_2**	**O_3**	**O_4**
c_1'	-1.4	-1.4	0	0
c_3'	0	0	-1.4	-1.4

Ilustración 3. Ejemplo de aplicación del LSA.

Supongamos que vamos a aplicar el análisis semántico latente a esos datos. Debemos indicar el número deseado de las dimensiones en la salida, en nuestro caso sabemos que vamos a querer dos dimensiones (dos características).

Nótese que el número de objetos sigue siendo sin cambios, es igual a cuatro. Los valores que corresponden a cada característica y cada objeto se han cambiado, sin embargo, dos nuevas características describen muy bien los cuatro objetos que tenemos y la eliminación de dos características no afecta su capacidad descriptiva.

En otro ejemplo vamos a considerar dos dimensiones, en cada una de las cuales se marca un valor. Se puede mapear esas dos dimensiones a otra dimensión, como se muestra en la Ilustración 4. Las proyecciones en la nueva dimensión siguen describiendo nuestros objetos.

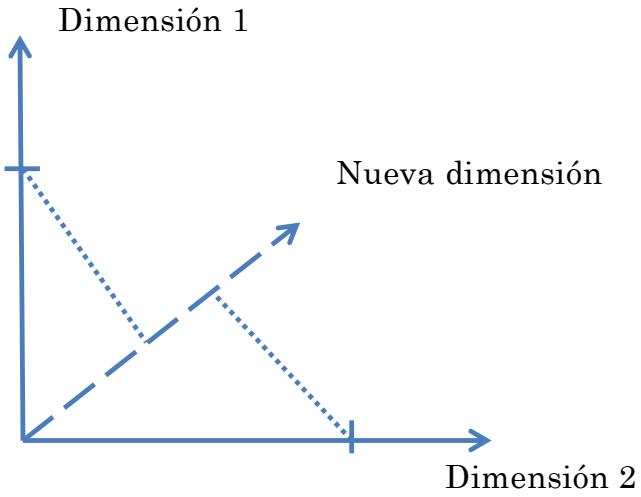


Ilustración 4. Proyecciones a una nueva dimensión.

Otra metáfora que podemos utilizar describiendo el análisis semántico latente es que debemos dar vuelta a nuestro espacio multidimensional tratando de mantener nuestros datos con menores cambios posibles, y en el mismo momento deshacerse de algunas dimensiones.

Uso del análisis semántico latente

Entonces, como hemos mencionado antes, el análisis semántico latente es una manera de reducir las dimensiones en un modelo de espacio vectorial. Se considera que el análisis semántico latente refleja las propiedades distribucionales de sus elementos, es decir, permite captar la similitud de los contextos de palabras o n-gramas, sin embargo, en la práctica es difícil que se presenten unos casos claros.

Al aplicar el análisis semántico latente se necesita indicar un parámetro importante: cuántas dimensiones debe tener el nuevo espacio vectorial, es decir, cuál es la reducción. Se recomienda tener valores de 100 a 500 dimensiones, aunque es cuestión de verificarlo experimentalmente para cada tarea.

Existen librerías que implementan el análisis semántico latente, por lo que recomendamos utilizar el código ya disponible libremente (C, Python, etc.). El análisis semántico latente es un procedimiento que consume algo de tiempo de procesamiento computacional, pero no es excesivamente largo.

Capítulo 5.
Diseño de experimentos en la lingüística computacional

Aprendizaje automático en la lingüística computacional

Como lo hemos mencionado antes en el libro, en el análisis automático de lenguaje natural (procesamiento de lenguaje natural, PLN) y en la lingüística computacional cada vez son más populares los métodos relacionados con el aprendizaje automático computacional (*machine learning*, en inglés). Aplicando estos métodos se obtienen resultados cada vez mejores [14, 19, 21, 23, 28, 36, 37, 42, 43, 45].

La intención principal detrás de la aplicación de los métodos de aprendizaje automático es tratar de modelar la formulación de hipótesis por parte de los

lingüistas y su posterior verificación. En este caso se sustituye la intuición humana por las grandes cantidades de datos textuales —posiblemente con marcas adicionales hechas manualmente—, y por los métodos sofisticados de aprendizaje que se basan en las matemáticas y en las estadísticas. De ninguna manera se pretende sustituir a los lingüistas en el proceso de investigación, sino desarrollar herramientas que puedan ser útiles para ellos.

Por otro lado, la aplicación de métodos de aprendizaje automático permite la evaluación exacta de las hipótesis sobre el lenguaje, la reproducción de los resultados y hasta cierto punto convierte a la lingüística computacional en una ciencia más exacta. En este sentido, algunas partes de la lingüística computacional ya requieren procedimientos empíricos, cuando la hipótesis formulada se verifica utilizando los experimentos computacionales basados en los datos, y no sólo en la intuición de los hablantes nativos o del propio experimentador.

Los métodos más utilizados en la lingüística computacional actual son en su mayoría los métodos de aprendizaje automático supervisado, es decir, se utilizan datos marcados manualmente para realizar entrenamiento. Otra posibilidad está relacionada con

el uso de los métodos no supervisados, cuando el propio sistema debe aprender directamente de los datos. Claro está que es mucho más difícil utilizar los métodos no supervisados, porque en este caso la propia computadora debe descubrir algo en los datos, sin alguna intervención humana. Normalmente para poder aplicar estos métodos se requiere una cantidad de datos realmente enorme.

Los métodos de aprendizaje automático específicos más utilizados en la lingüística computacional son Bayes ingenuo (*Naive Bayes*, *NB*), máquinas de vectores de soporte (*support vector machines*, *SVM*), clasificador basado en árboles de decisión *J48*.

Los conceptos básicos en el diseño de experimentos

Por lo mencionado anteriormente, en la lingüística computacional ya son aplicables los conceptos muy comunes en, digamos, recuperación de información [4] —de precisión y especificidad (*precision* y *recall*, en inglés)—, que miden la viabilidad de una hipótesis de manera formal. La combinación armónica de esas dos medidas se llama

la medida $F1$. Siempre recomendamos utilizar esta última para comparación de los métodos. Son conceptos relativamente sencillos. Suponemos que tenemos una colección de documentos y una petición. También tenemos un sistema que estamos evaluando. El sistema genera la respuesta, es decir, nos presenta algunos documentos recuperados considerados relevantes para la petición; vamos a llamarlos "todos recuperados".

Entre esos documentos algunos son recuperados correctamente, es decir, son documentos relevantes: "relevantes, recuperados"; mientras que otros son errores de recuperación cometidos por el sistema —son documentos "no relevantes, recuperados"—. Eso quiere decir que los documentos que llamamos "todos recuperados" consisten en "relevantes, recuperados" y "no relevantes, recuperados".

De aquí surge el concepto de precisión: ¿qué tan buena es la respuesta con respecto a sí misma? ¿Cuántos documentos en la respuesta son recuperados de manera correcta? Precisión (P) es la relación de "relevantes recuperados" con respecto a "todos recuperados".

$$P = \frac{relevantes\ recuperados}{todos\ recuperados}.$$

Por ejemplo, la precisión es 1, cuando todos los documentos en la respuesta son correctamente recuperados.

Existen, sin embargo, otros documentos que son relevantes para la petición, pero el sistema no los recuperó; vamos a llamarlos "relevantes, no recuperados". De aquí surge el concepto de especificidad (*recall, R*). ¿Qué tan buena (específica) fue la respuesta del sistema con respecto a la colección? ¿Pudo recuperar la mayoría de los documentos relevantes o solo unos pocos? Especificidad es la relación de los documentos "relevantes, recuperados" a todos los documentos relevantes ("todos relevantes"); este último conjunto incluye los documentos "relevantes, recuperados" y "relevantes, no recuperados".

$$R = \frac{relevantes\ recuperados}{todos\ relevantes}.$$

Especificidad es igual a 1 cuando el sistema recuperó todos los documentos relevantes. Siempre existe una relación entre precisión y especificidad: si tratamos de aumentar uno de esos valores, el otro disminuye.

Finalmente, la fórmula para la medida $F1$ que combina la precisión P y la especificidad R es:

$$F1 = \frac{2 \times P \times R}{P + R}.$$

La medida se llama $F1$ porque se da el mismo peso a la precisión y a la especificidad —igual a uno—. Se puede darles pesos diferentes, y se producen medidas relacionadas con $F1$ pero diferentes.

Otro concepto que se aplica en el diseño de experimentos es el llamado "línea base" (*baseline*), que corresponde a un método comúnmente aceptado del estado del arte que resuelve el mismo problema, y que debe ser superado por la hipótesis propuesta. Normalmente la línea base es un método no muy sofisticado. También es conveniente hacer una comparación del método que estamos proponiendo con otros métodos más sofisticados del estado de arte.

Dado que estamos hablando de datos marcados por los seres humanos (etiquetadores) de manera manual, que se usan para la verificación del método, se utiliza el concepto de "estándar de oro" (*gold standard*). Como los datos son marcados de manera manual, se supone que no tienen errores (de aquí viene la idea que es de "oro"), y si el sistema lo puede alcanzar, entonces el sistema funciona realmente muy bien.

Una cuestión interesante está relacionada con la concordancia entre los juicios humanos, es decir, si los propios seres humanos marcan de manera diferente algún fenómeno lingüístico, entonces no podemos esperar que una máquina resuelva correctamente este mismo fenómeno. Para medir la concordancia entre evaluadores se usa la "estadística kappa" [9].

También aquí se encuentra el concepto de la línea tope (*top line*, en inglés), que significa el valor máximo que puede ser alcanzado por un programa dado la falta de concordancia entre evaluadores. En general, es recomendable usar varios evaluadores, y no sólo uno, para tener juicios mejor fundamentados y menos sesgados.

Para realizar los experimentos se utiliza comúnmente la técnica llamada validación cruzada de k volúmenes (*k-fold cross validation*), donde k tiene un valor numérico, normalmente igual a 10. La técnica consiste en dividir todos los datos en 10 (o k) volúmenes. Primero se utiliza el volumen 1 para la evaluación de los resultados del sistema, y los volúmenes 2 a 10 para el entrenamiento, después se elige el volumen 2 para la evaluación, y los otros nueve volúmenes para el entrenamiento, y así sucesivamente.

De esta manera el sistema es evaluado 10 veces sobre datos diferentes y se entrena 10 veces sobre datos un poco diferentes, finalmente se toma el promedio de las 10 evaluaciones como el valor final. De esa manera se neutraliza la influencia de las fluctuaciones que pueden existir en los datos.

Nótese que es muy importante no realizar la evaluación utilizando los mismos datos sobre los cuales se hizo el entrenamiento: por eso es la división en *k* volúmenes. Si se usan los mismos datos, el algoritmo de aprendizaje puede detectar las características específicas de esos datos, en lugar de generalizar. Eso se llama "exceso de aprendizaje" (*overfitting*). El problema de exceso de aprendizaje siempre está presente en cualquier situación de aprendizaje y hay que tomar medidas para evitarlo.

Para realizar todos los procedimientos descritos, tenemos que representar el problema de manera formal. Repetimos brevemente algunas de las ideas que ya hemos discutido anteriormente. El modo más utilizado de representación de los objetos que estamos investigando es el modelo de espacio vectorial que hemos discutido en los capítulos anteriores. Se representa el problema como un problema de clasificación automática en un espacio, que es

precisamente el espacio vectorial. Los objetos se representan como conjuntos de valores de características, es decir, a cada objeto le corresponde un vector de dichos valores (de aquí el término "espacio vectorial"). Eso significa que cada objeto es un punto en el espacio multidimensional de características. Es muy fácil imaginar este modelo para el caso de dos características (dos dimensiones). Para un mayor número de dimensiones simplemente hay que suponer que es similar.

Después de definir el espacio, se define una métrica en este espacio. Normalmente es la métrica de similitud de objetos, definida por la similitud de coseno. La idea de dicha similitud es: entre más se parecen los dos objetos entre sí, menor es el ángulo entre los vectores que les correspondan en el espacio definido y, por lo tanto, mayor es el coseno de este ángulo.

Ahora bien, la siguiente pregunta es: ¿cómo elegir las características para definir el espacio vectorial? En este momento en nuestro proceso del diseño de experimento empiezan a prevalecer las consideraciones lingüísticas: es precisamente la parte lingüística que determina las características que podemos elegir.

Por ejemplo, la idea más sencilla utilizada en recuperación de información es utilizar todas las palabras en varios documentos como sus características, y después comparar esos documentos: cuantas más palabras "similares" tienen dos documentos, más se parecen entre sí estos documentos. Así se construye el espacio vectorial para la tarea de recuperación de información. Para medir "similitud" de las palabras se usa su *tf-idf*.

Obviamente en este caso, las ideas de qué tipo de información lingüística se puede utilizar se restringen por el requisito formal de utilizar el modelo de espacio vectorial, es decir, es nuestra obligación de representar los objetos como los conjuntos de valores de características.

La siguiente posibilidad más utilizada en la práctica, y que ya tiene cierto fundamento lingüístico, es la idea de utilizar n-gramas como características en el modelo de espacio vectorial. Como se señaló con anterioridad, el concepto de n-grama tradicional —las secuencias de palabras (u otros tipos de elementos) según aparecen en los textos—, tiene un fundamento lingüístico que consiste en incluir la información sintagmática de las palabras (seguir o preceder otras palabras).

Sin embargo, hubiera sido mucho más provechoso utilizar un conocimiento aún más "lingüístico", es decir, que involucre más información propiamente lingüística. Como un camino en esta dirección, en nuestros trabajos anteriores [46, 47, 48, 49, 50, 51] hemos propuesto un nuevo concepto de n-gramas, que contiene aún mayor cantidad de información de naturaleza lingüística que los n-gramas tradicionales: n-gramas sintácticos. La idea de los n-gramas sintácticos consiste en construirlos siguiendo la ruta en el árbol sintáctico. De esa manera los n-gramas sintácticos siguen siendo n-gramas, pero permiten introducir información sintáctica en los métodos de aprendizaje automático.

Los n-gramas sintácticos se discuten en la siguiente parte de este libro, al igual que algunas otras posibilidades de construcción de manera no lineal de n-gramas: n-gramas filtrados y n-gramas generalizados.

Diseño de experimentos

Después de considerar todos los puntos mencionados ya podemos describir el diseño de

experimentos en la lingüística computacional moderna que incluye los siguientes pasos:

1) Definir la tarea (por ejemplo, puede ser generación de resúmenes, detección de autoría, recuperación de información, etc.). A menudo la definición de una tarea diferente de las tareas estándar puede contener una aportación científica interesante.

2) Seleccionar los textos para el diseño de experimentos; lo que es equivalente a la preparación de un corpus. Se puede utilizar varios criterios para selección de textos. Siempre es mejor utilizar los corpus ya existentes para una tarea dada —eso permite una interpretación más objetiva de los resultados y la comparación. Sin embargo, no es obligatorio —siempre se puede desarrollar el corpus propio—. En este caso, recomendamos hacerlo público para que otras personas lo puedan utilizar.

3) Preparar el estándar de oro (*gold standard*). Para eso hay que marcar (etiquetar) manualmente los textos (o alguna parte de ellos). El tipo de marcación depende del problema que estamos resolviendo. Se recomienda basarse en los criterios de varios etiquetadores, es decir, que sean varias personas que lo hagan. También es recomendable calcular los

valores de concordancia entre ellos para determinar la línea tope.

4) Construir el modelo de espacio vectorial, seleccionando las características y sus valores. También recomendamos probar varios tipos de características: unigramas, n-gramas, n-gramas sintácticos, varios tipos de elementos que los compongan (palabras, lemas, etiquetas *POS*, etc.); y varios tipos de valores: *tf*, *tf-idf*, etc.

5) Definir e implementar uno o varios métodos de línea base —que son los métodos muy sencillos—; y también uno o varios métodos del estado del arte —que son ya métodos más complejos—. Claro está que todos esos métodos deben resolver el mismo problema.

6) Seleccionar o implementar uno o varios métodos de aprendizaje automático supervisado. Se recomienda utilizar los métodos ya hechos, por ejemplo, el sistema WEKA [25] implementa decenas de métodos de aprendizaje automático. Al mismo tiempo hay que analizar los parámetros de esos métodos y sus rangos de cambios, para poder posteriormente probar combinaciones de varios parámetros. Como ya hemos mencionado, los métodos más utilizados en la lingüística computacional son

Bayes ingenuo (*NB*), máquinas de vectores de soporte (*SVM*) y el clasificador basado en árboles de decisión *J48*. Sin embargo recomendamos probar la mayor cantidad posible de los métodos. También en este momento hay que considerar la posibilidad de aplicar el análisis semántico latente (o alguna transformación de espacio vectorial similar) para reducir la dimensionalidad del problema. Por ejemplo, WEKA tiene una implementación de *LSA*. En gran medida eso sería la aportación científica.

7) Convertir los datos textuales en el formato que aceptan los métodos de aprendizaje automático basándose en el modelo de espacio vectorial construido. En caso de WEKA son los archivos ARFF (*attribute relation file format*).

8) Realizar los experimentos de clasificación automática: esos son los procedimientos que pueden realizar los métodos de aprendizaje automático —ellos saben distinguir de manera automática entre varias clases definidos en el corpus basándose en el modelo de espacio vectorial construido—. Se recomienda utilizar el procedimiento de validación cruzada basada en 10 volúmenes. Los experimentos se realizan para el modelo propuesto y se comparan con

los resultados de los algoritmos de línea base y los algoritmos del estado del arte.

9) Calcular los valores de precisión y especificidad, y sobre todo la medida $F1$, para todos los métodos mencionados y realizar la comparación. Si el método propuesto obtiene mejores resultados, entonces este método es justificado.

En la situación actual, la aportación científica en gran medida consiste en construcción del modelo de espacio vectorial y algunos procedimientos adicionales que permiten transformar este modelo (el método específico propuesto por el investigador). Hasta cierto punto la aportación científica también puede incluir la definición del problema y el análisis de qué método de aprendizaje automático y reducción de dimensiones y con qué parámetros es mejor para el problema seleccionado.

Este es el paradigma actual de investigación. Esperamos que en el futuro más énfasis se dé al desarrollo de las características lingüísticas —de manera manual, automática o semiautomática—, como ya sucede en los modelos generativos basados en las características lingüísticas locales, tipo Campos Condicionales Aleatorios (*Conditional Random Fields*,

en inglés), que para muchas tareas dan mejores resultados que los métodos más tradicionales.

PARTE II.
CONSTRUCCIÓN NO LINEAL DE N-GRAMAS

Capítulo 6.
N-gramas sintácticos: el concepto

La idea de los n-gramas sintácticos

Como ya hemos mencionado, la idea principal de las características formales aplicables en la lingüística computacional se relaciona con el modelo de espacio vectorial y con el uso de los n-gramas como características en este espacio; lo que también incluye los unigramas, es decir, las palabras.

Recordemos que los n-gramas tradicionales son secuencias de elementos textuales (palabras, lemas, etiquetas gramaticales, etc.) según su orden de aparición en documentos. Los n-gramas tradicionales representan la información sintagmática y son ampliamente utilizados en varias tareas de la lingüística computacional con buenos resultados. Los n-gramas tradicionales ignoran la información sintáctica, se basan únicamente en la información

sintagmática, la relación establecida al "seguir otra palabra". La siguiente pregunta es, ¿cómo podemos seguir utilizando la técnica de n-gramas, de la cual sabemos da muy buenos resultados, y al mismo tiempo involucrar la información sintáctica? La respuesta que estamos proponiendo en este libro es la manera especial de obtención de n-gramas —la manera no lineal. Nuestra propuesta general es sacar los n-gramas siguiendo las rutas en árboles sintácticos.

Intuitivamente, seguimos teniendo n-gramas, pero logramos evitar el ruido introducido por la estructura superficial del lenguaje. Este tipo de ruido puede aparecer, porque en el nivel superficial las palabras no relacionadas sintácticamente pueden aparecer juntas. Podemos combatir este fenómeno si vamos a seguir las relaciones sintácticas reales que unen las palabras, aunque éstas no sean vecinos inmediatos.

Nótese que en este capítulo estamos proponiendo obtener un n-grama sintáctico como un fragmento de una ruta continua, no estamos considerando bifurcaciones en la ruta —más adelante presentaremos ejemplos de eso. En los capítulos posteriores vamos a presentar el concepto de n-

gramas sintácticos no continuos (el concepto más general), donde siguiendo la ruta en un árbol sintáctico se permite entrar en las bifurcaciones y regresar.

Entonces, por el momento vamos a continuar con la descripción de los n-gramas sintácticos que llamamos "continuos" para ilustrar el concepto de un n-grama sintáctico como tal. Los n-gramas son "continuos" en el sentido en que desde cualquier punto de la ruta se puede moverse exactamente a un punto siguiente.

Se puede observar que es precisamente una manera no lineal de construcción de n-gramas, dado que no tomamos sus elementos según su orden lineal. En este caso la idea del orden lineal se refiere al nivel superficial del texto, donde los elementos obligatoriamente tienen que seguir unos a otros —recordando a F. de Saussure, el carácter lineal del significante—.

A menudo con el término "n-grama sintáctico" quieren decir que es un n-grama compuesta por los etiquetas gramaticales (*POS*), por ejemplo, en [1]. Creemos que es el uso incorrecto de este término, porque las etiquetas gramaticales representan no la información sintáctica, sino la información

morfológica. Se usa la información sintáctica para la desambiguación de las etiquetas, pero eso no justifica su uso de esa manera.

Se ha propuesto una idea un poco parecida a la nuestra de usar el concepto de *skip*-gramas (salto-gramas), por ejemplo, véase [7]; es decir, formar secuencias de elementos de manera aleatoria saltando algunos de ellos. Claramente es la construcción no lineal, pero los n-gramas construidos de esa manera contienen más ruido que los n-gramas tradicionales, y su número se vuelve demasiado grande.

Una modificación de esa idea es usar no todos *skip*-gramas, sino solamente aquellos con mayores frecuencias [26], llamados "secuencias frecuentes maximales". Sin embargo, para la construcción de esas secuencias se necesitan algoritmos muy sofisticados, y seguimos con el problema de interpretación de ellas —la realidad lingüística que les corresponde no va más allá de la búsqueda de algunas combinaciones de palabras.

Otras ideas nuestras que estamos presentando en el libro de cómo construir los n-gramas de manera no lineal se relacionan con los conceptos de n-gramas filtrados —por ejemplo, usando su *tf-idf* como un filtro ANTES de construir n-gramas— y con n-gramas

generalizados. Digamos, para los n-gramas generalizados podemos utilizar siempre la primera palabra en una lista de sinónimos (*synset*), o promover las palabras en una ontología a conceptos más generales, sustituir las palabras por estos conceptos, y después construir los n-gramas de esos conceptos más generalizados.

Cabe mencionar que la idea de utilizar información estructural de relaciones de palabras en tareas específicas se ha presentado anteriormente, [3, 24, 32] sin embargo, en ninguno de estos trabajos se ha generalizado, ni se ha relacionado con la idea de los n-gramas.

El trabajo [41] ha propuesto una idea similar en el campo de análisis semántico, donde la utilidad de la información sintáctica se demuestra en tareas muy específicas de:

1. prioridad semántica (*semantic priming*), es decir, los experimentos psicolingüísticos sobre la similitud de palabras,
2. detección de sinónimos en las pruebas TOEFL, y
3. ordenamiento de sentidos de palabras según su importancia.

En nuestra opinión, este trabajo no tuvo mucha resonancia en otras tareas de PLN precisamente por

no relacionar la información sintáctica con los n-gramas, que es la herramienta principal en la gran mayoría de tareas, ni por mostrar su utilidad en otras tareas que no son tan específicamente orientadas a la semántica.

Para la ilustración del concepto de n-gramas sintácticos vamos a considerar, como ejemplo, dos oraciones tomadas de un libro de Julio Verne. El primer ejemplo está en el idioma inglés, y el segundo está en el idioma español. Para ambos ejemplos previamente construimos el árbol sintáctico, es decir, tenemos la información sintáctica representada en el formato de dependencias.

Ejemplo de sn-gramas continuos en español

Para analizar el ejemplo en español utilizamos el analizador sintáctico del sistema FreeLing [5, 39, 40]. Las relaciones sintácticas se muestran en el resultado de análisis con sangría, es decir, los bloques con la misma sangría tienen la misma palabra principal (si no aparece otra posible palabra principal entre esas palabras). La oración en español es la siguiente:

El doctor Ferguson se ocupaba desde hacía mucho tiempo de todos los pormenores de su expedición.

El analizador sintáctico FreeLing genera la siguiente salida en el formato de dependencias.

```
grup-verb/top/(ocupaba ocupar VMII1S0 -) [
  morfema-verbal/es/(se se P0000000 -)
  sn/subj/(doctor doctor NCMS000 -) [
    espec-ms/espec/(El el DA0MS0 -)
    w-ms/sn-mod/(Ferguson ferguson NP00000 -)
  ]
  prep/modnomatch/(desde desde SPS00 -)
  grup-verb/modnomatch/(hacía hacer VMII1S0 -) [
    sn/cc/(tiempo tiempo NCMS000 -) [
      espec-ms/espec/(mucho mucho DI0MS0 -)
      sp-de/sp-mod/(de de SPS00 -) [
        sn/obj-prep/(pormenores pormenor NCMP000 -) [
          espec-mp/espec/(todos todo DI0MP0 -) [
            j-mp/espec/(los el DA0MP0 -)
          ]
        ]
      ]
      sp-de/sp-mod/(de de SPS00 -) [
        sn/obj-prep/(expedición expedición NCFS000 -) [
          espec-fs/espec/(su su DP3CS0 -)
        ]
      ]
    ]
    F-term/term/(. . Fp -)
```

]
]

Presentamos el árbol correspondiente en las ilustraciones 5 y 6. Se puede observar que el analizador sintáctico se equivocó en algunas ocasiones. Por ejemplo, colocó los grupos de palabras “de su expedición” y “de todos los pormenores” como dependientes de la palabra “tiempo” en lugar del verbo “ocuparse”. Los analizadores sintácticos pueden cometer este tipo de errores con algunos tipos de ambigüedad sintáctica difícil de resolver de manera automática. Sin embargo, aun así, en muchos casos eso no afectaría significativamente el funcionamiento del sistema, dado que la gran mayoría de relaciones se establecieron correctamente y los errores no son tan graves que arruinen la estructura del árbol. En el árbol abajo indicamos los errores del analizador sintáctico con las líneas de puntos. Sin embargo, para el procesamiento automático, el formato de salida del analizador no es muy cómodo.

Hemos desarrollado un software que convierte el formato de salida de FreeLing en otro formato. En este caso el formato de salida es similar al formato de salida del analizador sintáctico de Stanford. Este software está disponible libremente en la página personal del autor del libro.

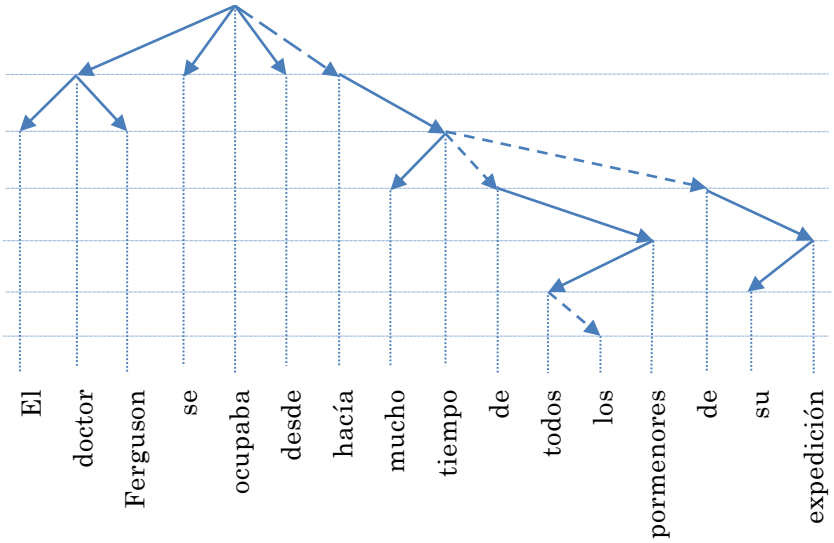


Ilustración 5. Ejemplo de un árbol sintáctico en español.

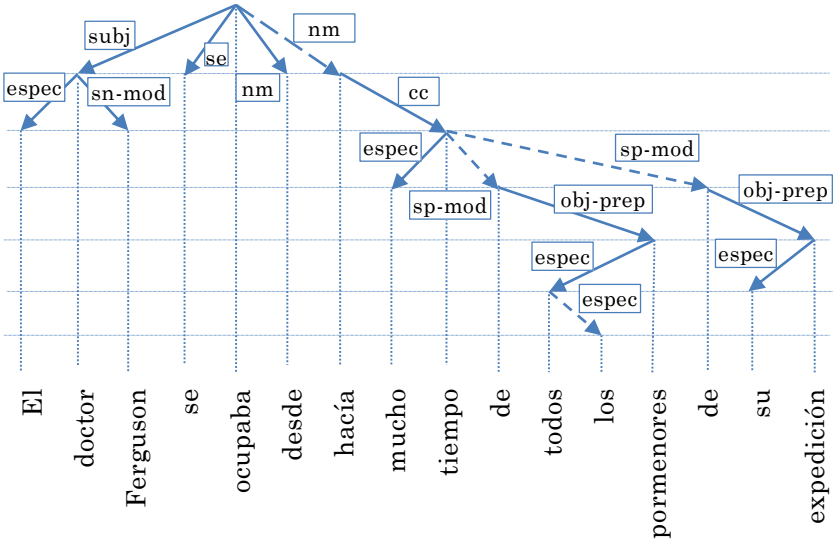


Ilustración 6. Ejemplo de un árbol sintáctico en español con etiquetas.

A continuación presentamos el resultado de la conversión de formato. Nótese que en este caso los números de palabras corresponden a las líneas de la salida generada por el analizador de FreeLing, y no a los números reales de su posición en la oración.

top(root-0, ocupaba-1)
es(ocupaba-1, se-2)
subj(ocupaba-1, doctor-3)
espec(doctor-3, el-4)
sn-mod(doctor-3, Ferguson-5)
modnomatch(ocupaba-1, desde-6)
modnomatch(ocupaba-1, hacía-7)
cc(hacía-7, tiempo-8)
espec(tiempo-8, mucho-9)
sp-mod(tiempo-8, de-10)
obj-prep(de-10, pormenores-11)
espec(pormenores-11, todos-12)
espec(todos-12, los-13)
sp-mod(tiempo-8, de-14)
obj-prep(de-14, expedición-15)
espec(expedición-15, su-16)

Utilizando este árbol podemos obtener los siguientes n-gramas sintácticos continuos.

Los bigramas son como sigue:

ocupaba se
ocupaba doctor
doctor el

doctor ferguson
ocupaba desde
ocupaba hacía
hacía tiempo
tiempo mucho
tiempo de
de pormenores
pormenores todos
todos los
tiempo de
de expedición
expedición su

Se obtienen los siguientes trigramas:

ocupaba doctor el
ocupaba doctor ferguson
ocupaba hacía tiempo
hacía tiempo mucho
hacía tiempo de
hacía tiempo de
tiempo de pormenores
de pormenores todos
pormenores todos los
tiempo de expedición
de expedición su

El número de 4-gramas es un poco menor:

ocupaba hacía tiempo mucho
ocupaba hacía tiempo de[1]
ocupaba hacía tiempo de
hacía tiempo de pormenores
hacía tiempo de expedición
tiempo de pormenores todos
de pormenores todos los
tiempo de expedición su

Existen pocos 5-gramas:

ocupaba hacía tiempo de pormenores
ocupaba hacía tiempo de expedición
hacía tiempo de pormenores todos
hacía tiempo de expedición su
tiempo de pormenores todos los

En este caso se puede obtener también tres 6-gramas:

ocupaba hacía tiempo de expedición su
ocupaba hacía tiempo de pormenores todos
hacía tiempo de pormenores todos los

[1] Este n-grama se repite porque en la oración existen dos palabras "de" diferentes. Eso significa que este n-grama tiene la frecuencia igual a dos en nuestra oración.

Y finalmente existe solo un 7-grama:

ocupaba hacía tiempo de pormenores todos los

La metodología de obtención de los n-gramas sintácticos continuos es similar para cualquier idioma. Como estamos observando es así para el caso de inglés y español, sin embargo, los árboles de dependencias tienen la misma estructura para todos los lenguajes.

Ejemplo de sn-gramas continuos en inglés

Para el ejemplo en inglés utilizamos el analizador sintáctico de Stanford (Stanford *parser*[2]) [8]. Primero presentamos la salida del analizador sintáctico, tal como lo genera el propio programa, y después en las ilustraciones 7 y 8 presentamos el árbol de dependencias utilizando flechas de varios niveles. El nivel de profundidad del árbol sintáctico es un concepto importante: empezamos de la raíz de la

[2] Un analizador sintáctico automático (*parser*) es un programa que construye los árboles sintácticos. Usualmente se basan en gramáticas formales de varios tipos.

oración y nos vamos bajando mientras seguimos la ruta sintáctica. La frase en inglés es como sigue.

The wildest cheering resounded on all sides; the name of Ferguson was in every mouth. (Lit.: "Un fuerte aplauso sonó por todos lados; el nombre de Ferguson estuvo en cada boca.")

La salida directa del analizador sintáctico consiste de dos partes. En la primera parte se presenta la información a nivel de constituyentes, donde una mayor sangría en la línea corresponde a una mayor profundidad de cada elemento. Sin embargo para nuestros propósitos más interés presenta la segunda parte de la salida que corresponde al modelo formal de dependencias, donde podemos observar las parejas de palabras y la relación sintáctica entre ellas. De esa segunda parte de la salida podemos construir directamente el árbol sintáctico, como se presenta en las ilustraciones 7 y 8.

Nótese que el analizador sintáctico automático a veces comete errores. Uno de estos errores lo marcamos en las ilustraciones con la línea de puntos, cuando la palabra *wildest* se tomó como el sustantivo, y la palabra *cheering* se tomó como el participio; realmente *cheering* es el sustantivo y *wildest* es el adjetivo.

Primera parte de la salida corresponde al formalismo de constituyentes:

```
(ROOT
 (S
  (S
   (NP
    (NP (DT The) (NN wildest))
    (VP (VBG cheering)))
   (VP (VBD resounded)
    (PP (IN on)
     (NP (DT all) (NNS sides)))))
  (: ;)
  (S
   (S
    (NP
     (NP (DT the) (NN name))
     (PP (IN of)
      (NP (NNP Ferguson))))
    (VP (VBD was)
     (PP (IN in)
      (NP (DT every) (NN mouth)))))
  (. .)))
```

Segunda parte de la salida corresponde al formalismo de dependencias:

```
det(wildest-2, The-1)
nsubj(resounded-4, wildest-2)
partmod(wildest-2, cheering-3)
root(ROOT-0, resounded-4)
prep(resounded-4, on-5)
```

det(sides-7, all-6)
pobj(on-5, sides-7)
det(name-10, the-9)
nsubj(was-13, name-10)
prep(name-10, of-11)
pobj(of-11, Ferguson-12)
parataxis(resounded-4, was-13)
prep(was-13, in-14)
det(mouth-16, every-15)
pobj(in-14, mouth-16)

Se puede observar que para cada par de palabras el analizador nos indica el tipo de la relación sintáctica existente entre ellas, y también el número de la palabra en la oración. Esa información es importante, porque en caso de que una palabra se repitiera en una oración, no tendríamos manera de determinar a cuál de las dos instancias de esa palabra se refiere el par dado.

En las ilustraciones 7 y 8 se presentan los mismos árboles sintácticos, siendo la única diferencia, que en la Ilustración 8 sobre cada flecha aparece el nombre de la relación sintáctica correspondiente.

Es bastante obvio después de nuestra explicación anterior, qué n-gramas es posible obtener de esta oración.

Por ejemplo, los siguientes bigramas pueden ser obtenidos:

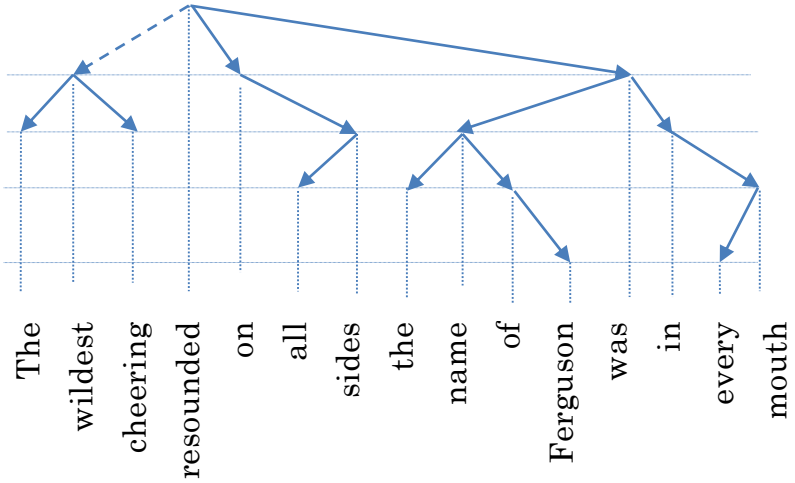


Ilustración 7. Ejemplo de un árbol sintáctico en inglés.

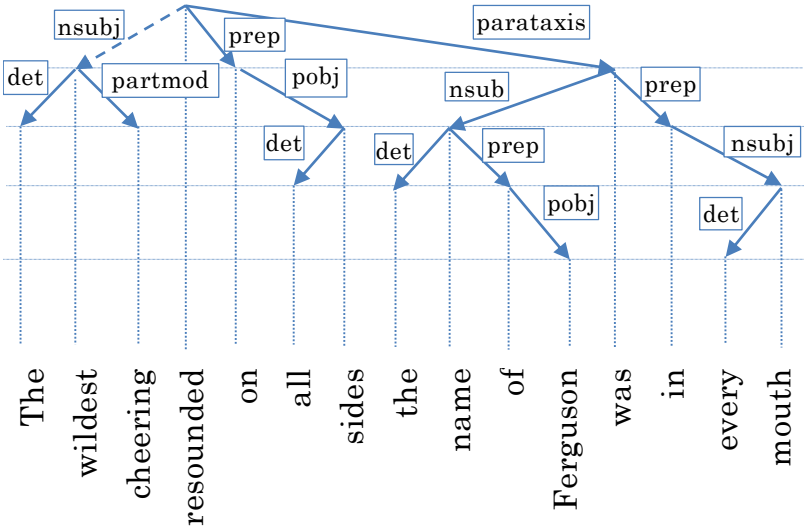


Ilustración 8. Ejemplo de un árbol sintáctico en inglés con etiquetas.

resounded wildest
wildest cheering
wildest the
resounded on
on sides
resounded was
was name
name of
of Ferguson
was in
in mouth
mouth every.

También los siguientes trigramas pueden ser extraídos:

resounded wildest the
resounded wildest cheering
resounded on sides
on sides all
resounded was name
resounded was in
was name the
was name of
name of ferguson
was in mouth
in mouth every.

Se extraen seis 4-gramas:

resounded on sides all
resounded was name the
resounded was name of
resounded was in mouth
was name of ferguson
was in mouth every.

Y finalmente existen solamente dos 5-gramas:

resounded was name of ferguson
resounded was in mouth every.

Capítulo 7.
Tipos de n-gramas sintácticos según sus componentes

N-gramas de elementos léxicos

Ahora bien, ya sabemos cómo obtener n-gramas sintácticos (aunque estamos considerando por el momento únicamente n-gramas sintácticos continuos), vamos a analizar qué tipos de n-gramas sintácticos existen según los elementos que pueden formarlos; es decir, qué tipo de elementos (componentes) pueden estar presentes como parte de n-gramas sintácticos. De hecho, las consideraciones que analizaremos son las mismas para cualquier tipo de n-gramas.

Obviamente la opción más directa son palabras, como sucede en todos los ejemplos mencionados

anteriormente. También está muy claro que en lugar de las palabras podemos utilizar sus formas normalizadas —los lemas—, que se obtienen después del procedimiento de normalización morfológica. Otra opción muy similar es utilizar las raíces unificadas de palabras (el proceso de obtención de las raíces se llama *stemming* en inglés). En este sentido el lema y la raíz unificada tienen la misma función: representan todo el conjunto de las formas gramaticales que corresponden a una palabra. La ventaja de utilizar la normalización morfológica es que se reduce el número de elementos que pueden componer los n-gramas, y de esa manera se reduce el número de n-gramas, es decir, son menos dimensiones en el modelo de espacio vectorial.

N-gramas de etiquetas gramaticales

De manera similar, en lugar de palabras podemos utilizar la información gramatical (etiquetas gramaticales) correspondiente a cada palabra, por ejemplo, para el caso del español podemos utilizar las etiquetas que producen los analizadores morfológicos o sintácticos, como FreeLing: NCFS000, VMII1S0, etc. El término correspondiente en inglés es etiquetas

POS (*POS tags*), o simplemente *POS*. Las etiquetas que se usan en FreeLing es un estándar de facto para la codificación de la información morfológica en el español, se llama el estándar EAGLES. En este caso, la primera letra corresponde a la clase gramatical: *N* es sustantivo (*noun*), *V* es verbo (*verb*), etc. La segunda letra refleja algunas propiedades léxicas, por ejemplo, en el caso de los sustantivos, *C* quiere decir que es un nombre común (otro valor de esta característica sería *P* que corresponde al nombre propio), etc. Las siguientes letras ya reflejan las características gramaticales, por ejemplo en caso de los sustantivos *F* corresponde al género femenino y *M* al género masculino, etc.

N-gramas de etiquetas de relaciones sintácticas

En caso de los n-gramas sintácticos también tenemos una nueva posibilidad de elementos en comparación con los n-gramas tradicionales: utilizar como elementos de n-gramas los nombres de relaciones sintácticas que están presentes en el árbol sintáctico, por ejemplo, *nsubj* o *pobj*, en caso de inglés (Ilustración 8) o *espec* o *obj-prep* para el español

(Ilustración 6). Podemos llamarlos etiquetas de relaciones sintácticas o en inglés *SR-tags* (*tags of syntactic relations*).

Los nombres de relaciones sintácticas están presentes en las ilustraciones mencionadas sobre las flechas. En este caso, un n-grama sería una secuencia de etiquetas *SR* como por ejemplo, *sp-mod obj-prep espec;* que en este caso es un trigrama.

N-gramas de caracteres

La última posibilidad que existe para los n-gramas tradicionales es utilizar los caracteres como sus elementos. Por ejemplo, en la frase *Juan lee*, se encuentran los siguientes bigramas: *"ju", "ua", "an", "n ", " l", "nl", "le", "ee"*. En este caso utilizamos el espacio entre palabras como el elemento de n-gramas, también se puede utilizar los signos de puntuación. Sin embargo, para algunas tareas es mejor no considerar los caracteres auxiliares.

También se puede utilizar el mismo concepto de caracteres como elementos para los n-gramas sintácticos: sacarlos de la misma manera que para los n-gramas tradicionales, pero primero sacar los bigramas o trigramas sintácticos de palabras. Y

después considerar la secuencia de palabras en un n-grama sintáctico como fuente para la construcción de los n-gramas de caracteres. Es cuestión de futuras investigaciones determinar si este tipo de n-gramas sintácticos es útil. Resulta que la aplicación de los n-gramas tradicionales de caracteres presenta buenos resultados en algunas tareas, por ejemplo, en la detección de autoría [52]. Sin embargo, desde nuestro punto de vista, la aplicación de los n-gramas de caracteres es hasta cierto punto anti-intuitivo y es necesario analizar las razones de su funcionamiento aceptable, véase la sección de n-gramas filtrados de caracteres más adelante.

N-gramas mixtos

Finalmente, pueden existir los n-gramas mixtos. Eso quiere decir que algunos elementos de un n-grama pueden ser de un tipo, y otros elementos del mismo n-grama pueden ser de otro tipo. Parece que los caracteres no pueden participar en construcción de los n-gramas mixtos, porque su naturaleza es diferente: los otros tipos de elementos representan palabras, mientras que los caracteres representan partes de palabras.

Respecto a los n-gramas mixtos, se deberá analizar en el futuro qué combinaciones de los elementos —palabras, etiquetas *POS* (*POS-tags*), etiquetas *SR* (*SR-tags*)— y en qué posiciones —al inicio, en medio de un n-grama, o al final— están dando mejores resultados.

Clasificación de n-gramas según sus componentes

Resumiendo, podemos decir que existen n-gramas sintácticos de:

- Elementos léxicos (palabras, lemas, o raíces),
- Etiquetas de las categorías gramaticales (*POS tags*),
- Nombres de las relaciones sintácticas (*SR tags*),
- Caracteres,
- N-gramas sintácticos mixtos (combinaciones de los tipos anteriores).

En [41] se menciona la idea de ponderar las relaciones entre los elementos de un n-grama sintáctico. Esa idea no nos parece aplicable directamente en el contexto de modelos de espacio

vectorial, donde los n-gramas son las características (dimensiones del espacio). Sin embargo, esa idea puede ser útil en el momento de calcular los pesos de n-gramas sintácticos, aparte de los valores tradicionales de *tf-idf*.

Capítulo 8.
N-gramas sintácticos continuos y no continuos

N-gramas sintácticos continuos

En los capítulos anteriores hemos introducido el nuevo concepto de los n-gramas sintácticos, es decir, n-gramas obtenidos siguiendo las rutas en un árbol sintáctico.

Como lo mostraremos en los siguientes capítulos, los n-gramas sintácticos pueden dar mejores resultados que los n-gramas tradicionales en algunas tareas de PLN. Nótese que los n-gramas sintácticos pueden ser utilizados en cualquier tipo de problemas donde se utilizan los n-gramas tradicionales porque permiten la construcción del espacio vectorial. Nosotros analizaremos su aplicación para el problema de detección de autoría.

La desventaja de los n-gramas sintácticos consiste en el hecho requerir el análisis sintáctico automático previo, lo que toma cierto tiempo de procesamiento, aunque no es una limitación muy seria. Otra limitación es que no para todos los idiomas existen analizadores sintácticos automáticos, pero sí para los idiomas con mayor presencia en el mundo, como el español o el inglés.

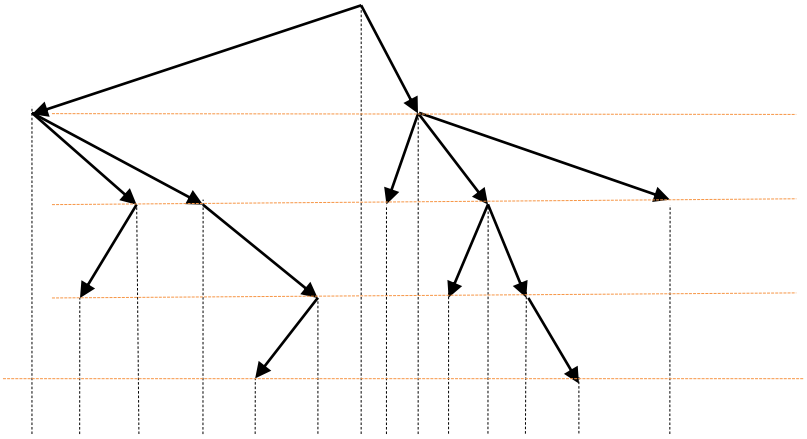
A continuación en este capítulo vamos a comparar los n-gramas sintácticos continuos con los n-gramas sintácticos n- continuos.

Un árbol sintáctico de una frase muestra[3] se presenta en las ilustraciones 9 y 10, utilizando el formalismo de dependencias y constituyentes [15, 22, 51]. Note que la expresión *de_mala_gana* se considera como una sola palabra.

Independientemente del tipo de elementos que constituyen los n-gramas sintácticos, todos los n-gramas sintácticos considerados en los capítulos anteriores, son continuos. Eso quiere decir que la ruta sintáctica que estamos siguiendo nunca tiene bifurcaciones. Por ejemplo, la ruta marcada con flechas en negrita en la Ilustración 11 corresponde a

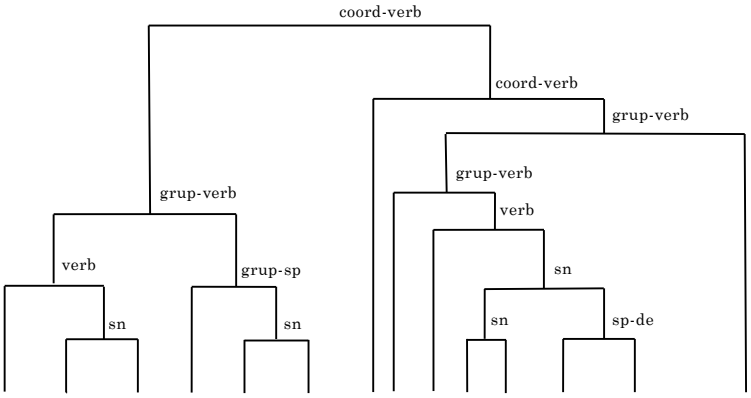
[3]Frase de una de las obras de A. Conan-Doyle.

un 5-grama sintáctico continuo. En este caso es el 5-grama: "*y di par de vueltas*".



Tomé el pingajo en mis manos y le di un par de vueltas de_mala_gana

Ilustración 9. Árbol sintáctico (analizado por FreeLing) representado con dependencias.



Tomé el pingajo en mis manos y le di un par de vueltas de_mala_gana

Ilustración 10. Árbol sintáctico (analizado por FreeLing) representado con constituyentes.

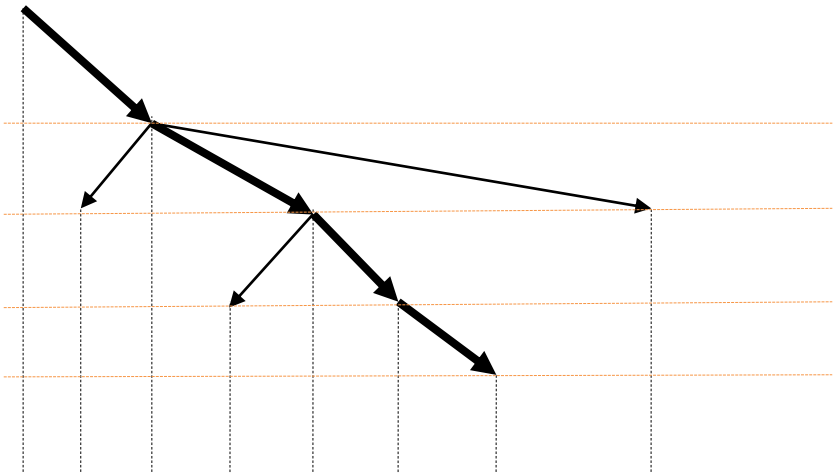


Ilustración 11. N-gramas sintácticos continuos en el fragmento del árbol sintáctico: un 5-grama.

A continuación presentaremos otro tipo de n-gramas sintácticos, donde se permiten las bifurcaciones.

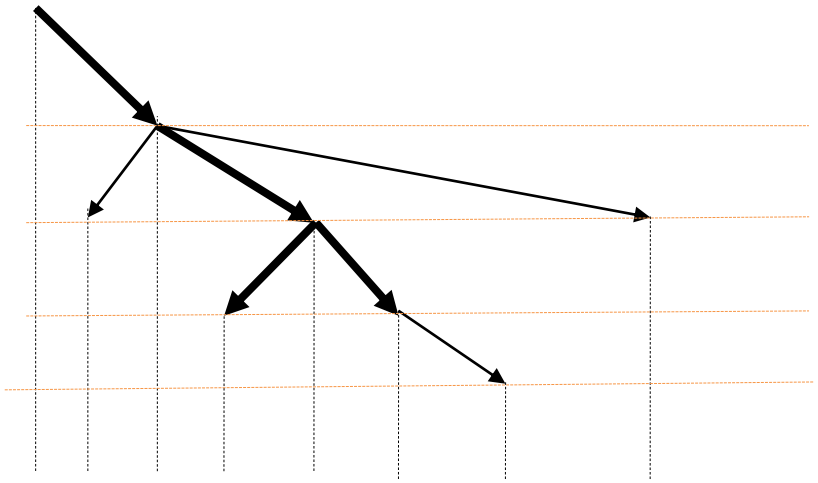
N-gramas sintácticos no continuos

En esta sección vamos a presentar una generalización del concepto de n-gramas sintácticos continuos: los n-gramas sintácticos no continuos [50, 51].

Como se puede observar en la sección anterior, la intuición detrás del concepto de n-gramas sintácticos

continuos está relacionada principalmente con el hecho de que una secuencia de palabras relacionadas puede ser considerada como tal, en su totalidad.

Sin embargo, existen otros conceptos lingüísticos interesantes, que no caben en el modelo de una secuencia unidimensional, digamos, las valencias verbales (o patrones de rección).



y le di un par de vueltas de_mala_gana

Ilustración 12. N-gramas sintácticos no continuos en el fragmento del árbol sintáctico: un 5-grama.

Por ejemplo, el verbo *comprar* tiene los actantes: *quién, qué, de quién, por cuanto dinero*. Sería muy interesante tenerlos presentes al mismo tiempo en un n-grama.

Sin embargo, tanto para el caso de n-gramas tradicionales, como de n-gramas sintácticos continuos, todos esos componentes hubieran sido separados en n-gramas diferentes. Entonces, la intuición detrás del concepto de n-gramas sintácticos no continuos es precisamente tratar de unir las palabras relacionadas semánticamente, aunque éstas no tengan una ruta continua, pero sí cuenten con alguna ruta que las conecte.

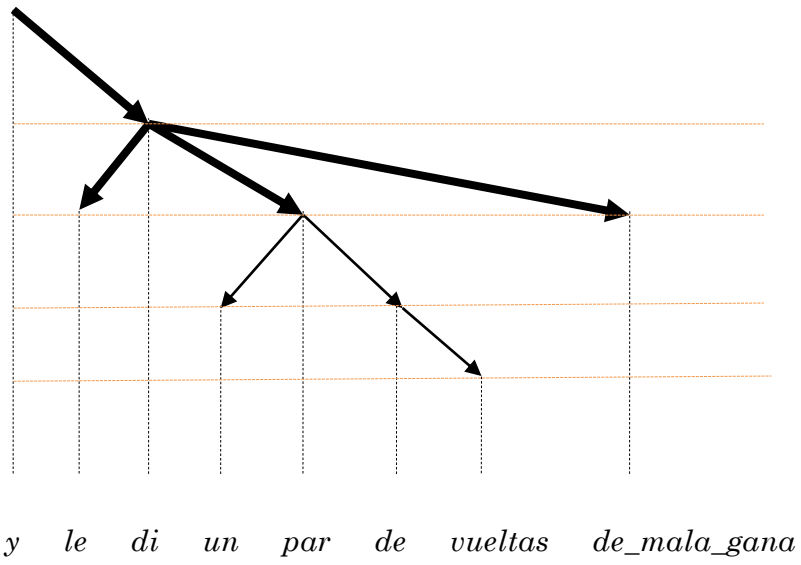


Ilustración 13. N-gramas sintácticos no continuos en el fragmento del árbol sintáctico: otro 5-grama.

Es muy fácil dar una definición formal de n-gramas sintácticos no continuos: son todos los sub-árboles de longitud *n* de un árbol sintáctico.

En las ilustraciones 12 y 13 se presentan dos ejemplos de los n-gramas sintácticos no continuos que son fragmentos de la frase considerada anteriormente. Los n-gramas están marcados con las flechas en negrita.

En el primer caso (Ilustración 12) es un 5-grama "*y le di par* [*un*, *de*]". Note que es necesario introducir un metalenguaje para la representación de los n-gramas sintácticos no continuos para resolver la posible ambigüedad. Este metalenguaje se discute en el siguiente capítulo.

En el segundo caso (Ilustración 13) el 5-grama es "*y di* [*le*, *par*, *de_mala_gana*]".

Entonces, de manera formal, los n-gramas sintácticos continuos se definen como todos los sub-árboles sin bifurcaciones de longitud *n* de un árbol sintáctico. Otro modo de decirlo de manera formal es que cada punto de la ruta está conectado con un solo punto.

Eso quiere decir que los n-gramas sintácticos continuos son un caso particular de n-gramas sintácticos no continuos, al aplicar las definiciones

propuestas. La longitud de un árbol es el número de arcos en este árbol, lo que corresponde al valor de *n* (en caso de los n-gramas).

Otro término que podemos proponer para denotar los n-gramas sintácticos no continuos, es t-n-gramas (*tree n-grams*, en inglés) es decir, n-gramas de árboles. La sugerencia de A. Gelbukh es usar el término "t-gramas, árbol-gramas" (*tree grams*, *t-grams*, en inglés), sin embargo nos parece un poco mejor el término "n-gramas de árboles", dado que así se establece la relación del término propuesto con el concepto muy tradicional de los n-gramas. Otra posibilidad es usar el término "n-gramas arbóreos". Del otro lado, una consideración a favor del término "t-grama" es su forma más simple. Sin embargo, siempre preferimos quedarnos con el término "n-grama sintáctico".

Es cuestión de futuros trabajos determinar qué tipo de n-gramas: continuos o no continuos, son mejores para varios tipos de tareas de la lingüística computacional. Es posible que para algunos tipos de tareas son mejores unos, y para otros tipos de tareas, otros.

Cabe mencionar que el número de n-gramas sintácticos no continuos es mayor que el de los n-

gramas sintácticos continuos, dado que los últimos son un caso particular de los primeros.

El algoritmo de construcción (u obtención) de los n-gramas sintácticos no continuos es relativamente sencillo. Para el nodo raíz hay que considerar todas las posibles combinaciones de sus hijos con el tamaño no mayor que *n*, y repetir este procedimiento de manera recursiva para cada nodo hijo. Y así de manera sucesiva hay que pasar por todos los nodos del árbol sintáctico.

Capítulo 9.
Metalenguaje de representación de n-gramas sintácticos

Surge la pregunta ¿cómo representar (simbolizar) a los n-gramas sintácticos no continuos sin utilizar su forma gráfica? Cabe recordar que los n-gramas sintácticos continuos son simplemente secuencias de palabras (tomados siguiendo la ruta en un árbol sintáctico), pero el caso de los n-gramas sintácticos no continuos es diferente.

Estamos proponiendo utilizar los siguientes convenios. Nótese que son convenios, por lo que pueden ser modificados en un futuro. Dentro de cada n-grama sintáctico no continuo pueden existir unas partes continuas y una o varias bifurcaciones. Vamos a separar los elementos continuos de n-gramas con espacios en blanco —nada más—, y en la parte de la

bifurcación vamos a poner comas, además vamos a usar paréntesis para marcar la parte de bifurcaciones, para evitar que posteriormente aparezca la ambigüedad estructural.

Nótese que siempre hemos considerado que el elemento de nivel más alto aparece del lado izquierdo (la palabra principal de la relación), y el componente de nivel más bajo del lado derecho (la palabra dependiente de la relación sintáctica). Es la manera más natural, pero es necesario mencionarlo de manera explícita.

Dos ejemplos de los 5-gramas sintácticos no continuos se presentan en las ilustraciones 12 y 13: "*y di par [un, de]*", "*y di [le, par, de_mala_gana]*".

Nótese que no podemos evitar el uso ni de los corchetes ni de las comas en nuestro metalenguaje, porque aparece la ambigüedad. En nuestros ejemplos pareciera ser que podemos evitar el uso de los corchetes, si la coma indica que la palabra anterior es parte de una bifurcación. Es así solo en caso de que los elementos en la bifurcación no contienen una ruta. Por ejemplo, en caso de un 5-grama "*y di* [*par un*, *de_mala_gana*]", *un* depende de *par*, lo que se expresa con el espacio en blanco, pero en este caso claramente no podemos evitar el corchete.

Es importante mencionar que los corchetes y las comas son ahora parte de los n-gramas, pero eso de ninguna manera impide la identificación de n-gramas sintácticos que sean iguales. Aunque tengan algunos símbolos adicionales a las palabras, pueden compararse sin problemas.

Otro ejemplo de una posible ambigüedad. Consideremos un caso en el que un n-grama tenga dos bifurcaciones y varios fragmentos continuos. Por ejemplo, los n-gramas "*a [b, c [d, e, f]]*" y "*a [b, c [d, e], f]*" tienen un nodo —*f*— como un tercer nodo debajo del nodo *c*, o bien como un tercer nodo debajo del nodo *a,* véase la Ilustración 14.

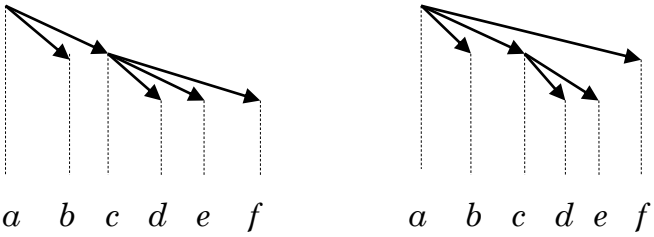


Ilustración 14. Ejemplo de una posible ambigüedad en bifurcaciones.

Ahora bien, tenemos dos posibilidades de manejar las partes en bifurcaciones, es decir, las partes separadas por comas:

1. tal como aparecen en el texto, que es la manera más natural, o

2. ordenarlos de alguna manera, por ejemplo, alfabéticamente, lo que disminuye el número de n-gramas.

Esta última opción nos permite tomar en cuenta los cambios relacionados con el orden de palabras. Sin embargo, se necesitan más investigaciones para determinar cuál de las dos opciones es mejor y para qué tarea de PLN.

Otra posibilidad que nos gustaría mencionar es marcar directamente dentro de los n-gramas sintácticos no continuos su profundidad. La intuición detrás de esta idea es que para algunos tipos de n-gramas sintácticos no continuos puede ser importante su posición en el árbol sintáctico de la oración. En este caso la notación sería: "*y*$_1$ *di*$_2$ *par*$_3$ *[un*$_4$*, de*$_4$*]*", "y$_1$ *di*$_2$ *[le*$_3$*, par*$_3$*, de_mala_gana*$_3$*]*". Técnicamente, sería suficiente marcar el nivel de la primera palabra únicamente; podemos marcar los demás niveles también, pero no es estrictamente necesario.

Note que esa notación también puede servir para denotar la estructura de un n-grama, sustituyendo las comas y los corchetes. Sólo se debe tomarse en cuenta que todos los n-gramas deben empezar con el nivel "1" —independientemente de su nivel real en la oración—, dado que en el caso contrario no será

posible identificar los n-gramas iguales pero pertenecientes a diferentes niveles. Tomando en cuenta esta complicación en su construcción nos inclinamos al uso de los corchetes y las comas.

A continuación vamos a considerar dos ejemplos de construcción de n-gramas sintácticos no continuos, uno para el español y otro para el inglés, y los comparamos con los n-gramas sintácticos continuos.

Capítulo 10.
Ejemplos de construcción de n-gramas sintácticos no continuos

Ejemplo para el español

En esta sección vamos a presentar ejemplos de la construcción de n-gramas sintácticos continuos y no continuos para el español. Tomemos la frase muestra del capítulo anterior:

Tomé el pingajo en mis manos y le di un par de vueltas de mala gana.

Para construir automáticamente los n-gramas sintácticos, es necesario aplicar antes un programa de análisis sintáctico automático, llamado en inglés *parser*. Para el español hemos utilizado el programa

FreeLing [5, 39, 40], que está disponible de manera gratuita.

El analizador sintáctico puede construir el árbol utilizando dos formatos: constituyentes y dependencias. El árbol de dependencias se presentó en la Ilustración 9, y el de constituyentes en la Ilustración 10. Ambos formatos tienen esencialmente la misma información de relaciones de palabras. Para los fines de construcción de los n-gramas sintácticos nos parece mejor utilizar las dependencias, porque esta representación es más transparente. Sin embargo, de igual manera se puede utilizar el árbol de constituyentes.

Cabe mencionar que el analizador sintáctico primero realiza el análisis morfológico y la lematización. Como se puede observar, a cada palabra de la oración le corresponde su lema y la información gramatical, por ejemplo, "*Tomé tomar* VMIS1S0". Primero viene la palabra, después el lema, y al final la información gramatical.

Ya hemos mencionado que la información gramatical utiliza el esquema de codificación EAGLES, que es el estándar de facto para el análisis morfológico automático del español. Por ejemplo, en la etiqueta VMIS1S0, la primera letra "V" corresponde

al “verbo” (“N” hubiera sido un sustantivo, “A” un adjetivo, etc.), la “I” es el indicativo, la “S” significa el pasado, “1” es la primera persona, la otra letra “S” significa “singular”. Como se puede observar, en cada posición se codifica un tipo específico de la información gramatical, y cada etiqueta tiene como máximo siete posiciones, de las cuales algunos pueden no usarse en algunos casos, por ejemplo, en caso de los sustantivos.

Primero presentamos los resultados de análisis automático de la oración utilizando el formalismo de constituyentes (Ilustración 10).

```
+coor-vb_[
  grup-verb_[
    +verb_[
      +(Tomé tomar VMIS1S0 -)
    ]
    sn_[
      espec-ms_[
        +j-ms_[
          +(el el DA0MS0 -)
        ]
      ]
      +grup-nom-ms_[
        +n-ms_[
```

```
        +(pingajo pingajo NCMS000 -)
       ]
      ]
     ]
     grup-sp_[
      +prep_[
       +(en en SPS00 -)
      ]
      sn_[
       espec-fp_[
        +pos-fp_[
         +(mis mi DP1CPS -)
        ]
       ]
       +grup-nom-fp_[
        +n-fp_[
         +(manos mano NCFP000 -)
        ]
       ]
      ]
     ]
    ]
    +(y y CC -)
    grup-verb_[
     patons_[
```

```
  +paton-s_[
    +(le le PP3CSD00 -)
  ]
]
+grup-verb_[
  +verb_[
    +(di dar VMIS1S0 -)
  ]
]
sn_[
  espec-ms_[
    +indef-ms_[
      +(un uno DI0MS0 -)
    ]
  ]
  +grup-nom-ms_[
    +n-ms_[
      +(par par NCMS000 -)
    ]
  ]
  sp-de_[
    +(de de SPS00 -)
    sn_[
      +grup-nom-fp_[
        +n-fp_[
```

```
            +(vueltas vuelta NCFP000 -)
          ]
        ]
      ]
    ]
  ]
  sadv_[
    +(de_mala_gana de_mala_gana RG -)
  ]
 ]
 F-term_[
  +(. . Fp -)
 ]
]
```

Información muy similar se presenta utilizando el formalismo de dependencias de la Ilustración 9.

```
coor-vb/top/(y y CC -) [
 grup-verb/co-v/(Tomé tomar VMIS1S0 -) [
  sn/dobj/(pingajo pingajo NCMS000 -) [
   espec-ms/espec/(el el DA0MS0 -)
  ]
  grup-sp/sp-obj/(en en SPS00 -) [
   sn/obj-prep/(manos mano NCFP000 -) [
    espec-fp/espec/(mis mi DP1CPS -)
   ]
```

```
  ]
 ]
 grup-verb/co-v/(di dar VMIS1S0 -) [
  patons/iobj/(le le PP3CSD00 -)
  sn/dobj/(par par NCMS000 -) [
   espec-ms/espec/(un uno DI0MS0 -)
   sp-de/sp-mod/(de de SPS00 -) [
    sn/obj-prep/(vueltas vuelta NCFP000 -)
   ]
  ]
  sadv/cc/(de_mala_gana de_mala_gana RG -)
 ]
 F-term/modnomatch/(. . Fp -)
]
```

Como ya mencionamos es más sencillo utilizar las dependencias, porque ellas prácticamente ya contienen los n-gramas sintácticos.

Se puede observar que las tres palabras *de_mala_gana* realmente representan un solo adverbio.

Ahora bien, vamos a presentar los ejemplos de los n-gramas sintácticos extraídos. Primero presentamos los n-gramas sintácticos continuos.

Los bigramas sintácticos (en principio, en bigramas no hay diferencia entre bigramas continuos y no continuos) son:

y tomé
tomé pingajo
pingajo el
tomé en
en manos
manos mis
y di
di le
di par
par un
par de
de vueltas
di de_mala_gana.

Los trigramas continuos son:

y tomé pingajo
y tomé en
tomé pingajo el
tomé en manos
en manos mis
y di le
y di par
y di de_mala_gana

di par un
di par de
par de vueltas.

Los 4-gramas continuos son:

y tomé pingajo el
y tomé en manos
tomé en manos mis
y di par un
y di par de
di par de vueltas

Al presentar los n-gramas sintácticos no continuos, no vamos a repetir los mismos elementos —n-gramas continuos—, aunque ellos también forman parte de los n-gramas sintácticos no continuos. Nótese que en este caso tenemos que usar la notación propuesta para los n-gramas no continuos para poder distinguirlos de otras configuraciones posibles. La notación forma parte de los n-gramas, no es algo adicional, es el propio n-grama. Entonces, los trigramas no continuos nuevos en comparación con los n-gramas continuos son:

tomé [pingajo en]
di [le par]
di [le de_mala_gana]

di [par de_mala_gana]
par [un de]

Los 4-gramas no continuos nuevos son:

tomé [pingajo el, en]
tomé [pingajo, en manos]
di [le, par un]
di [le, par de]
di [le, par, de_mala_gana]
di [par un, de_mala_gana]
di [par de, de_mala_gana]
par [un, de vueltas]

Ejemplo para el inglés

En esta sección vamos a analizar la construcción de n-gramas sintácticos para el inglés. Para simplificar la comparación con el español, tomaremos la traducción de la misma frase que en la sección anterior. Nótese que en este caso la imagen correspondiente al árbol se generó de manera automática; el código que lo permite hacer está disponible en la página personal del autor[4].

[4] http://www.cic.ipn.mx/~sidorov

I took the scrap in my hands and turned it a couple of times unwillingly.

Se puede utilizar el mismo analizador sintáctico que en los ejemplos anteriores, es decir, FreeLing, sin embargo, vamos a probar con otro analizador sintáctico para el inglés que también hemos mencionado antes —analizador sintáctico de Stanford (Stanford *parser*) [15]—. El árbol de constituyentes es como se presenta en la Ilustración 15.

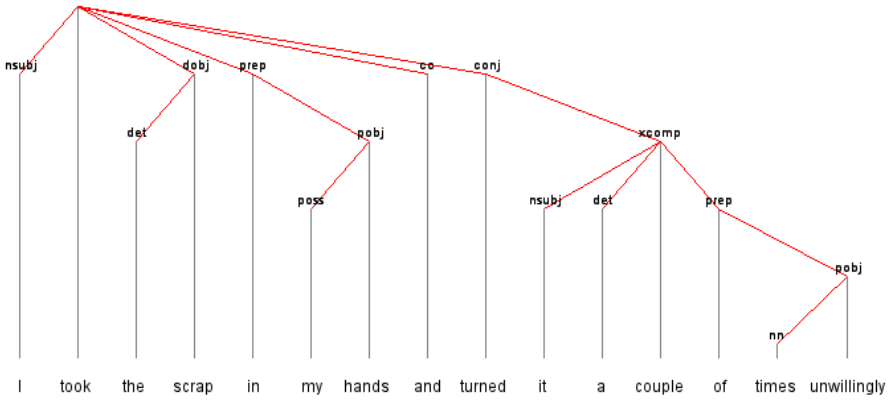


Ilustración 15. El árbol sintáctico del ejemplo para el inglés.

Como ya hemos mencionado, la mayoría de los analizadores sintácticos generan su salida tanto en el formalismo de dependencias como de constituyentes. La salida del formato de constituyentes es la siguiente.

```
(ROOT
  (S
    (NP (PRP I))
    (VP
      (VP (VBD took)
        (NP (DT the) (NN scrap))
        (PP (IN in)
          (NP (PRP$ my) (NNS hands))))
      (CC and)
      (VP (VBD turned)
        (S
          (NP (PRP it))
          (NP
            (NP (DT a) (NN couple))
            (PP (IN of)
              (NP (NNS times) (NN unwillingly)))))))
    (. .)))
```

Como ya se dijo antes el analizador sintáctico de Stanford usa para el árbol de dependencias una representación muy simple, pero expresiva: nombre de la relación, dos palabras —o sus *POS tags*, o lemas dependiendo de qué tipo de elementos queremos considerar— junto con sus números de posiciones en la oración. Primero se menciona la palabra principal y después la palabra dependiente, es decir, el orden de las palabras es importante. Esta información permite construir el árbol sintáctico de manera única. A

continuación presentamos la salida del analizador en el formato mencionado.

nsubj(took-2, I-1)
root(ROOT-0, took-2)
det(scrap-4, the-3)
dobj(took-2, scrap-4)
prep(took-2, in-5)
poss(hands-7, my-6)
pobj(in-5, hands-7)
cc(took-2, and-8)
conj(took-2, turned-9)
nsubj(couple-12, it-10)
det(couple-12, a-11)
xcomp(turned-9, couple-12)
prep(couple-12, of-13)
nn(unwillingly-15, times-14)
pobj(of-13, unwillingly-15)

Se puede observar que aunque la frase es muy parecida a la anterior, el otro analizador sintáctico aplicó otras reglas, específicamente, manejó de manera diferente la conjunción y también cometió algunos errores: por ejemplo, con la palabra *unwillingly*, relacionándola con *of* y no con *turned*; con la palabra *it*, relacionándola con *couple* y no con *turned*. Sin embargo, los errores de un analizador sintáctico no afectan de manera conceptual nuestra discusión, ya que nuestra tarea no consiste en

mejorar algún analizador sintáctico, sino utilizar las herramientas existentes; también vale la pena mencionar que eventualmente los analizadores sintácticos están mejorando. Otra consideración interesante que ya hemos mencionado es que los errores que cometen los analizadores sintácticos se relacionan usualmente con varios tipos de ambigüedad sintáctica.

Ahora procedamos a la construcción de los n-gramas sintácticos continuos y no continuos. Los bigramas sintácticos —recordemos que en principio no hay diferencia entre los bigramas sintácticos continuos y no continuos— son:

took I
took scrap
scrap the
took in
in hands
hands my
took and
took turned
turned couple
couple it
couple a
couple of

of unwillingly
unwillingly times.

Los trigramas sintácticos continuos son:

took scrap the
took in hands
in hands my
took turned couple
turned couple it
turned couple a
turned couple of
couple of unwillingly
of unwillingly times.

Los 4-gramas sintácticos continuos son:

took in hands my
took turned couple it
took turned couple a
took turned couple of
turned couple of unwillingly
couple of unwillingly times.

Como en el ejemplo anterior no vamos a repetir los mismos elementos, aunque los n-gramas continuos también forman parte de los n-gramas sintácticos no continuos.

Los trigramas sintácticos no continuos —nuevos— son (puede ser que algunos de ellos son errores del analizador sintáctico, eso no afecta la idea propuesta dado que son errores de otra naturaleza, que se puede corregir mejorando el propio analizador sintáctico):

took [I, scrap]
took [I, in]
took [I, and]
took [I, turned]
took [scrap, in]
took [scrap, and]
took [scrap, turned]
took [in, and]
took [in, turned]
took [and, turned]
couple [it, a]
couple [it, of]
couple [a, of]

Los 4-gramas no continuos (nuevos) son:

took [I, scrap the]
took [in, scrap the]
took [and, scrap the]
took [turned, scrap the]

took [I, in hands]
took [scrap, in hands]
took [and, in hands]
took [turned, in hands]
took [I, scrap, in]
took [I, scrap, and]
took [I, scrap, turned]
took [scrap, in, and]
took [scrap, in, turned]
took [in, and, turned]
couple[it, a, of]
couple[it, of unwillingly]
couple[a, of unwillingly]

Nótese que en este caso hemos tomado los elementos de los n-gramas sintácticos no continuos en su orden de aparición en el texto. Como mencionamos antes, otra opción es ordenarlos de alguna manera, por ejemplo, alfabéticamente.

Capítulo 11.
Análisis automático de autoría usando n-gramas sintácticos

Preparación del corpus para la detección automática de autoría

Hemos desarrollado algunos experimentos [46] para probar la utilidad del concepto de n-gramas sintácticos. Esencialmente, consideramos la tarea de atribución de autoría, es decir, tenemos textos de los cuales conocemos sus autores, y tenemos un texto para el cual debemos determinar su autor (entre los autores considerados solamente). En nuestro caso usamos un corpus de textos para tres autores.

La tarea de atribución de autoría es claramente una tarea de clasificación: los autores corresponden a nombres de las clases, las clases son los textos cuyo

autor se conoce, y la tarea consiste con base en un texto decidir en qué clase él debe estar [2, 29, 33, 52]. Utilizamos como características los n-gramas tradicionales y los n-gramas sintácticos de tamaños de dos a cinco. Utilizamos una herramienta que permite fácilmente aplicar varios algoritmos de clasificación disponibles: el sistema WEKA [25].

Utilizamos un corpus compuesto de obras de los autores: *Booth Tarkington, George Vaizey, Louis Tracy* (escribían en inglés en el siglo XIX). Nuestro corpus fue compuesto por cinco novelas de cada autor para el entrenamiento, en total 11 MB, y de tres obras de cada autor para la clasificación, en total 6 MB [46, 47, 48]. Para la obtención de los n-gramas sintácticos usamos el analizador sintáctico de Stanford.

Evaluación de detección de autoría con n-gramas sintácticos

Para el diseño de los experimentos utilizamos perfiles de varios tamaños. El término "perfil" quiere decir que utilizamos el número correspondiente de los n-gramas más frecuentes, por ejemplo, para el perfil de 400, utilizamos nada más 400 n-gramas que tienen mayor frecuencia en el corpus de entrenamiento, etc.

Utilizamos un algoritmo de clasificación estándar que se llama "máquinas de vectores de soporte" (SVM, *support vector machines*, en inglés). Se sabe que para muchas tareas el algoritmo de máquinas de vectores de soporte da resultados mejores que otros algoritmos de clasificación.

Cuadro 2. Resultados de atribución de autoría para bigramas.

Tamaño de perfil	**n-gramas sintácticos de etiquetas *SR***	**n-gramas de etiquetas *POS***	**n-gramas de caracteres**	**n-gramas de palabras**
400	100%	90%	90%	86%
1,000	100%	95%	95%	86%
4,000	100%	ND	90%	86%
7,000	100%	ND	ND	86%
11,000	100%	ND	ND	89%

Los resultados de clasificación (atribución de autoría) se presentan en el Cuadro 2 para los bigramas, y en el Cuadro 3 para los trigramas. Para comparar nuestros resultados, elegimos como otras maneras de selección de características n-gramas de

palabras, n-gramas de etiquetas *POS* y n-gramas de caracteres.

Cuadro 3. Resultados de atribución de autoría para trigramas.

Tamaño de perfil	**n-gramas sintácticos de etiquetas *SR***	**n-gramas de etiquetas *POS***	**n-gramas de caracteres**	**n-gramas de palabras**
400	100%	90%	76%	81%
1,000	100%	90%	86%	71%
4,000	100%	100%	95%	95%
7,000	100%	100%	90%	90%
11,000	100%	95%	100%	90%

La leyenda "ND" quiere decir que no se encontraron tantos n-gramas para el perfil específico, es decir, el número de n-gramas fue relativamente pequeño.

Como se puede observar, el método basado en los n-gramas sintácticos obtuvo los mejores resultados. Sin embargo, cabe mencionar que el problema que estamos considerando tiene la línea tope (*top line,* facilidad de obtener los resultados) bastante alto, dado que tenemos muchos datos de entrenamiento y

solamente estamos utilizando tres clases (autores). La información más detallada de los experimentos, y los datos de experimentos para mayor número de autores está disponible en nuestros trabajos previos [46, 47, 48], por lo que no presentamos en este libro una descripción detallada.

Capítulo 12.
N-gramas filtrados

Idea de n-gramas filtrados

En este capítulo y en el siguiente estamos presentando dos ideas relacionadas con la construcción no lineal de los n-gramas. Recordamos que la construcción no lineal consiste en que estamos tomando los elementos que van a formar n-gramas en el orden diferente de la representación superficial (textual); es decir, de alguna manera diferente de como las palabras (lemas, etiquetas *POS*, etc.) aparecen en el texto.

En los capítulos anteriores hemos discutido el concepto de n-gramas sintácticos, cuando el orden en que tomamos las palabras se define utilizando un árbol sintáctico.

Otra opción de obtener los n-gramas distintos de como parecen los elementos en el texto es filtrar

algunos elementos en el texto. De esa forma las palabras que no son vecinas pueden estar consideradas juntas.

De hecho, en su variante más sencillo, esa idea se aplica muy ampliamente —se filtran las palabras auxiliares (en inglés, *stop words*) durante la construcción de n-gramas.

Entonces, lo que estamos proponiendo en este capítulo, es aplicar esa idea de manera consistente y filtrar las palabras en textos utilizando algunos criterios. El criterio más obvio es usar la medida *tf-idf*, que se discutió ampliamente en la primera parte de este libro. Se pueden aplicar otras medidas. Algo muy parecido sucede con las palabras auxiliares, cuyos valores *tf-idf* son muy bajos, debido a su bajo *idf*.

El siguiente paso consiste en elegir un umbral para filtrar palabras. Es cuestión de experimentos futuros determinar los umbrales óptimos. Nótese que recomendamos considerar no sólo un umbral que corte la parte de arriba o la parte de abajo, sino la combinación de dos umbrales que al mismo tiempo corten una parte de arriba y otra parte de abajo. Eso corresponde a la intuición que los valores en medio son más importantes. Podemos generalizar esta idea y

utilizar no sólo dos umbrales sino una serie de pares de umbrales, aunque parece poco probable que existen los intervalos óptimos muy específicos para los valores de las características.

Nótese que esta idea puede ser fácilmente combinada con la idea de n-gramas sintácticos: simplemente debemos saltar en nuestra ruta en el árbol sintáctico las palabras filtradas según los umbrales que elegimos. En este caso una pregunta interesante es qué hacer en caso de bifurcaciones. Nosotros creemos que debemos seguir todas las posibles rutas que salen de una bifurcación, aunque no vamos a tomar en cuenta la palabra misma en esa bifurcación.

Ejemplo de n-gramas filtrados

Consideremos un simple ejemplo de construcción de los n-gramas filtrados. Suponemos que tenemos la frase que ya hemos utilizado como ejemplo.

Tomé el pingajo en mis manos y le di un par de vueltas de mala gana.

Y suponemos que queremos obtener n-gramas tradicionales construidos con palabras filtradas.

Vamos a presentar un cuadro con valores *tf-idf* de cada palabra en una colección imaginaria, ver el Cuadro 4.

Cuadro 4. Posibles valores *tf-idf* de palabras.

Palabra	***tf-idf***
manos	1.5
de_mala_gana	1.46
umbral de arriba, <1.3	
vueltas	**1.23**
tomé	**1.2**
par	**0.9**
pingajo	**0.7**
umbral de abajo, >0.1	
di	0.003
el	0
en	0
mis	0
y	0
le	0
un	0
de	0

Ahora, suponemos que vamos a filtrar las palabras con el valor de *tf-idf* muy bajo y muy alto. Por ejemplo, seleccionamos los umbrales: 1) ">0.1" y

2) “<1.3”. Sólo las palabras marcadas en negrita *tomé, pingajo, manos, par, vueltas* están en el intervalo que queremos considerar.

Nótese que los umbrales pueden tener el signo de comparación diferente de nuestro ejemplo, es decir, para el umbral de arriba usamos “menos, <”; se puede probar “más, >”, y utilizar las palabras que estén arriba del umbral, y no de abajo, como en el ejemplo.

Utilizando las palabras que quedaron en la consideración podemos construir los n-gramas. Por ejemplo, los bigramas tradicionales son:

tomé pingajo
pingajo manos
manos par
par vueltas.

Los bigramas sintácticos continuos serían:

tomé pingajo
tomé manos
par vueltas.

En caso de considerar los bigramas sintácticos no continuos se agrega el bigrama:

[tomé, pingajo].

Es curioso, porque hemos afirmado antes que los bigramas continuos y no continuos coinciden. Esa

situación cambia si empezamos a filtrar los elementos del árbol sintáctico sin eliminar las rutas. Para verificar cómo se construye este último bigrama, hay que volver a ver el árbol en la Ilustración 9 (pág. 107). Empezamos con la raíz que se filtra con base en los umbrales. Como estamos en una bifurcación, la marcamos con corchetes, y después encontramos dos elementos no filtrados a cada lado de la bifurcación, que separamos con coma. Pero una situación así solo puede presentarse cuando hacemos uso del filtrado.

N-gramas filtrados de caracteres

Otra idea que nos gustaría presentar en el libro está relacionada con construcción no lineal de n-gramas de caracteres. En este caso una posibilidad es primero filtrar las palabras —por ejemplo, utilizando *tf-idf*—, y después construir los n-gramas de caracteres de las palabras restantes. Esta idea es similar a la idea ya discutida en este capítulo.

Sin embargo, existe otra posibilidad de cómo filtrar los caracteres antes de que empecemos a construir los n-gramas de ellos. Básicamente la idea es filtrar los caracteres que ocupan ciertas posiciones en las palabras o tienen ciertas características.

Más específicamente, con respecto a las características, podemos filtrar, por ejemplo, las vocales y construir n-gramas del resto de caracteres. Se pueden probar varios tipos de características de caracteres para varias tareas.

Si se trata de posiciones, podemos, por ejemplo, solamente considerar los caracteres que se encuentran, digamos, a tres símbolos del inicio o del fin de cada palabra, e ignorar (filtrar) los caracteres que están en el resto de la palabra. Esa estrategia de filtrado en el idioma español debe tomar en cuenta más los afijos (sufijos, prefijos, flexiones) y dar menor peso a los n-gramas que representan las raíces de palabras. Cabe mencionar, que podemos hacerlo al revés: considerar los n-gramas de caracteres que corresponden a las raíces y tratar de descartar los n-gramas que están relacionados con los elementos gramaticales.

Se sabe que los n-gramas de caracteres dan buenos resultados en la tarea de detección de autoría, es decir, reflejan el estilo personal. La razón de este hecho no está muy clara, por eso creemos que los experimentos con varias estrategias de construcción no lineal de n-gramas de caracteres pueden aclarar qué fenómeno gramatical o léxico está detrás de los n-

gramas de caracteres. Es necesario realizar experimentos con varios parámetros y varias estrategias de construcción de los n-gramas.

Capítulo 13.
N-gramas generalizados

Idea de n-gramas generalizados

Otra idea relacionada con la construcción no lineal de n-gramas —es decir, utilizar los elementos distintos o en el orden distinto de su aparición en textos— es la idea de sustituir palabras por sus sinónimos o por los conceptos generalizados que les corresponden según alguna ontología.

Vamos a llamar este tipo de n-gramas "n-gramas generalizados". Su construcción es no lineal porque los n-gramas no contienen las palabras según su aparición en textos.

Cuando estamos sustituyendo palabras por un sinónimo, estamos en el mismo nivel de una ontología; y cuando estamos utilizando sus hiperónimos, estamos subiendo en una ontología. Para ambas situaciones es muy deseable realizar primero la

desambiguación de los sentidos de palabras, porque la selección correcta de los sinónimos e hiperónimos depende de eso en gran medida.

La idea general detrás de los n-gramas generalizados es disminuir la variedad léxica de textos, de esa manera se disminuye sustancialmente el número de los n-gramas. Es muy claro que esa idea puede combinarse fácilmente con la idea de los n-gramas sintácticos y de los n-gramas filtrados.

La aplicación de información de los sinónimos es muy sencilla: simplemente para cada palabra tomamos su lista de sinónimos, por ejemplo, *synsets* de WordNet o utilizamos cualquier diccionario de sinónimos, y sustituimos esa palabra por el primer sinónimo en esta lista; después procedemos a construir los n-gramas.

En caso de utilizar hiperónimos disponibles en ontologías pueden existir varias estrategias. Podemos utilizar siempre el nivel actual de cada palabra más una constante.

$$nivel\_hiperónimo = nivel\_de\_palabra + c.$$

En este caso debemos subir c niveles en la ontología que estamos utilizando.

Otra posibilidad es fijar algún nivel razonablemente bajo en la ontología, y siempre subir

hasta este nivel. Si nos encontramos con una palabra de nivel muy alto, es recomendable dejarla como está y no bajar al nivel correspondiente.

Entonces, de esa manera sustituimos las palabras por sus sinónimos o hiperónimos, y después podemos construir los n-gramas de esos nuevos elementos.

Ejemplo de n-gramas generalizados

El caso de sinónimos nos parece muy obvio. Vamos a considerar un ejemplo con hiperónimos.

Usemos la misma frase del capítulo anterior como ejemplo. Suponemos que también utilizamos el filtrado, y vamos a quedar con las palabras filtradas: *tomé, pingajo, manos, par, vueltas*.

Como ejemplo vamos a utilizar la estrategia subir un solo nivel en la ontología.

Suponemos que tenemos los siguientes hiperónimos: *manos* → *brazos* (es un holónimo, estrictamente hablando, pero nos sirve), *tomé* → *actué*, *pingajo* → *herramienta*, *vuelta* → *movimiento*, *par* → *número*. Obviamente esa información depende de la ontología específica, donde están las palabras y correspondencias que podemos encontrar.

Ahora podemos construir los n-gramas utilizando los conceptos generalizados, y no palabras como tales, por ejemplo, el bigrama "*tomé pingajo*" será sustituida por "*actué herramienta*" o "*actuar herramienta*". Depende de nuestros propósitos si este cambio nos sirve o no. La ventaja es que las palabras "*martillo, cuchillo*..." tendrían el mismo hiperónimo "*herramienta*", igual que las palabras "*tomé, clavé, corté*..." tendrían el hiperónimo "*actúe*" o "*actuar*".

Es cuestión de futuros estudios para determinar la utilidad de los n-gramas generalizados y las tareas donde se obtienen mejores resultados con su aplicación.

Bibliografía

1. Agarwal, A., Biads, F., Mckeown, K.R.: Contextual Phrase-Level Polarity Analysis using Lexical Affect Scoring and Syntactic N-grams. Proceedings of the 12th Conference of the European Chapter of the ACL (EACL), pp. 24–32 (2009)
2. Argamon, S., Juola, P.: Overview of the international authorship identification competition at PAN-2011. In: Proc. of 5th Int. Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (2011)
3. Baayen, H., Tweedie, F. and Halteren, H.: Outside The Cave of Shadows: Using Syntactic Annotation to Enhance Authorship Attribution. Literary and Linguistic Computing, pp. 121–131 (1996)
4. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)
5. Carreras, X., Chao, I., Padró, L., Padró, M.: FreeLing: An Open-Source Suite of Language Analyzers. In: Proceedings of the 4th International

Conference on Language Resources and Evaluation (LREC'04) (2004)

6. Cazés Menache, D., del Castillo, N., Mansilla, R., Pineda, L.A., Sidorov, G., Sierra Martínez, G.: El dominio de la lingüística: más allá de las ciencias exactas y naturales. UNAM, 213 p. (2009)
7. Cheng, W., Greaves, C., Warren, M.: From n-gram to skipgram to concgram. International Journal of Corpus Linguistics 11, no. 4, pp 411-433 (2006)
8. de Marneffe, M.C., MacCartney, B., Manning, C.D.: Generating Typed Dependency Parses from Phrase Structure Parses. In: Proc. of LREC (2006)
9. Díaz Rangel, I., Sidorov, G., Suárez-Guerra, S.: Creación y evaluación de un diccionario marcado con emociones y ponderado para el español. Onomazein, 29 (2014)
10. Dumais, S.T.: Latent Semantic Analysis. Annual Review of Information Science and Technology 38: 188 (2005)
11. Fillmore, Ch., Langendoen, T. (eds): Studies in Linguistic Semantics. (1971)
12. Firth, J.R.: A Synopsis of Linguistic Theory. In: Palmer, F.R. (ed), (1968) Selected Papers of J.R. Firth 1952-59. London/Harlow: Longmans (1957)

13. Gale, W.A., Church, K.W.: A program for Aligning Sentences in Bilingual Corpora. In: Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics, Berkeley, California (1991)

14. Gelbukh, A., Alexandrov, M., Han, SangYong: Detecting Inflection Patterns in Natural Language by Minimization of Morphological Model. In: A. Sanfeliu, J.F. Martínez Trinidad, J.A. Carrasco Ochoa (Eds.) Lecture Notes in Computer Science N 3287, Springer-Verlag, pp. 432–438 (2004)

15. Gelbukh, A., Calvo, H., Torres, S.: Transforming a Constituency Treebank into a Dependency Treebank. Procesamiento de Lenguaje Natural, No 35. Sociedad Española para el Procesamiento de Lenguaje Natural (SEPLN) (2005)

16. Gelbukh, A., Sidorov, G.: Procesamiento automático del español con enfoque en recursos léxicos grandes. IPN, 307 p. (2010)

17. Gelbukh, A., Sidorov, G.: Approach to construction of automatic morphological analysis systems for inflective languages with little effort. Lecture Notes in Computer Science, N 2588, Springer-Verlag, pp. 215–220 (2003)

18. Gelbukh, A., Sidorov, G.: Zipf and Heaps Laws' Coefficients Depend on Language. Lecture Notes in Computer Science N 2004, Sringer-Verlag, pp. 330–333 (2001)
19. Gelbukh A., Sidorov, G.: Alignment of Paragraphs in Bilingual Texts using Bilingual Dictionaries and Dynamic Programming. Lecture Notes in Computer Science, N 4225, Springer-Verlag, pp 824-833 (2006)
20. Gelbukh, A., Sidorov, G., Guzman-Arenas, A.: Use of a weighted topic hierarchy for text retrieval and classification. Lecture Notes in Artificial Intelligence, No. 1692, Springer, pp. 130–135 (1999)
21. Gelbukh, A., Sidorov, G., Han, SangYong: On Some Optimization Heuristics for Lesk-Like WSD Algorithms. Lecture Notes in Computer Science, N 3513, Springer-Verlag, pp. 402–405 (2005)
22. Gelbukh, A., Sidorov, G., Han, SangYong, Hernández-Rubio, E.: Automatic Enrichment of Very Large Dictionary of Word Combinations on the Basis of Dependency Formalism. Lecture Notes in Artificial Intelligence N 2972, Springer-Verlag, pp 430-437 (2004)
23. Goldsmith, J.: Unsupervised Learning of the Morphology of a Natural Language. Computational Linguistics 27:2, 153-198 (2001)

24. Habash, N.: The Use of a Structural N-gram Language Model in Generation-Heavy Hybrid Machine Translation. LNCS, 3123, pp. 61–69 (2004)

25. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update; SIGKDD Explorations, 11(1) (2009)

26. Hernández-Reyes, E., Martínez-Trinidad, J. Fco., Carrasco-Ochoa, J.A., García-Hernández, R.A.: Document Representation Based on Maximal Frequent Sequence Sets. LNCS 4225, pp. 854–863 (2006)

27. Inteligencia Artificial. G. Sidorov (Ed.), Alfa Omega, (2014)

28. Jiménez-Salazar, H., Pinto, D., Rosso, P.: Uso del punto de transición en la selección de términos índice para agrupamiento de textos cortos. Procesamiento del Lenguaje Natural, 35, pp. 383-390 (2005)

29. Juola, P.: Authorship Attribution. Foundations and Trends in Information Retrieval. 1(3):233–334 (2006)

30. Jurafsky, D., Martin, J.: Speech and Language Processing. Prentice Hall (2009)

31. Kay, M., Roscheisen, M.: Text-translation alignment. Computational Linguistics, 19(1):121–142 (1993)
32. Khalilov, M., Fonollosa, J.A.R.: N-gram-based Statistical Machine Translation versus Syntax Augmented Machine Translation: comparison and system combination. In: Proceedings of the 12th Conference of the European Chapter of the ACL, pp. 424–432 (2009)
33. Koppel, M., Schler, J., Argamon, S.: Authorship attribution in the wild. Language Resources and Evaluation 45(1):83–94 (2011)
34. Lesk, M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. Proc. of ACM SIGDOC Conference, Toronto, Canada, pp. 24-26 (1986)
35. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA (1999)
36. Medina Urrea, A.: Automatic Discovery of Affixes by means of a Corpus: A Catalog of Spanish Affixes. Journal of Quantitative Linguistics 7(2), pp. 97–114 (2000)
37. Miranda-Jimenez, S., Gelbukh, A., Sidorov, G: Generación de resúmenes por medio de síntesis de

grafos conceptuales. Revista "SIGNOS. Estudios de Lingüística", 47(86) (2014)

38. Montes y Gómez, M., Gelbukh, A., López López, A., Baeza-Yates, R.: Flexible Comparison of Conceptual Graphs. Lecture Notes in Computer Science N 2113, Springer-Verlag, pp.102-111 (2001)
39. Padró, L., Collado, M., Reese, S., Lloberes, M., Castellón, I.: FreeLing 2.1: Five Years of Open-Source Language Processing Tools. In: Proceedings of 7th Language Resources and Evaluation Conference (LREC 2010), ELRA La Valletta, Malta (2010)
40. Padró, L., Stanilovsky, E.: FreeLing 3.0: Towards Wider Multilinguality. In: Proceedings of the Language Resources and Evaluation Conference (LREC 2012), ELRA, Turkey (2012)
41. Pado, S., Lapata, M.: Dependency-based construction of semantic space models. Computational Linguistics, 33(2): 161–199 (2007)
42. Pichardo-Lagunas, O., Sidorov, G., Cruz-Cortés, N., Gelbukh, A.: Detección automática de primitivas semánticas en diccionarios explicativos con algoritmos bioinspirados. Onomazein, 28 (2013)
43. Reyes, J.A., Montes, A., González, J.G., Pinto, D.E.: Clasificación de roles semánticos usando

características sintácticas, semánticas y contextuales. Computación y sistemas, 17(2): 263–272 (2013)

44. Sierra, G., Alarcón, R.: Recurrent patterns in definitory context. In: Proc. CICLing-2002, Computational Linguistics and Intelligent Text Processing. Lecture Notes in Computer Science N 2276, Springer-Verlag, pp. 438–440 (2002)

45. Sierra, G., McNaught, J.: Natural Language System for Terminological Information Retrieval. Lecture Notes in Computer Science, N 2588, Springer, pp. 543–554 (2003)

46. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L.: Syntactic Dependency-based N-grams as Classification Features. LNAI, 7630, pp. 1–11 (2012)

47. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L.: Syntactic Dependency-Based N-grams: More Evidence of Usefulness in Classification. LNCS, 7816 (Proc. of CICLing), pp. 13–24 (2013)

48. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L.: Syntactic N-grams as Machine Learning Features for Natural Language

Processing. Expert Systems with Applications, 41(3): 853–860 (2014)

49. Sidorov, G.: N-gramas sintácticos y su uso en la lingüística computacional. Vectores de investigación, 6(6): 1–15 (2013)
50. Sidorov, G.: Non-continuous syntactic n-grams. Polibits, 48: 67–75 (2013)
51. Sidorov, G.: Syntactic Dependency Based N-grams in Rule Based Automatic English as Second Language Grammar Correction. International Journal of Computational Linguistics and Applications, 4(2): 169–188 (2013)
52. Stamatatos, E.: A survey of modern authorship attribution methods. Journal of the American Society for information Science and Technology 60(3): 538–556 (2009)

Construcción no lineal de n-gramas en la lingüística computacional
Grigori Sidorov

Impreso por:
Kronos Digital S.A. de C.V.
Esparta No 2, Col. Alamos, CP 03400,
Del. Benito Juarez, México DF

Noviembre 2013
Edición 500 ejemplares