Facultad de Ingeniería de la Universidad de Buenos Aires

Ingeniería en Informática

Tesis de Grado

Aplicación de la teoría de Agentes al modelo de grafos para la detección de patrones en Textos

Tesista: FEDERICO, Fernando Carlos
Padrón: 83195
E-mail: fernandofederico1984@yahoo.com.ar

Director: Dr. Juan M. Ale

- Abril de 2008 -

Resumen

Text Mining puede ser definido como el descubrimiento de conocimiento en grandes colecciones de textos. Se asocia principalmente al descubrimiento de patrones interesantes como clusters, asociaciones, desviaciones, similitudes, y diferencias. Por otro lado, los Attributed Relational Graphs (ARG) se definen como una extensión de los grafos ordinarios asociando atributos discretos o reales a sus vértices y arcos. El uso de los atributos permite a los ARG ser posibles de no sólo modelar estructuras topológicas de una entidad sino también sus propiedades no estructurales, que usualmente se pueden representar como vectores. Estas características hacen a esta herramienta un elemento útil a la hora de realizar búsqueda de patrones. En este trabajo, se define un algoritmo basado en grafos para la detección de patrones de textos. Debido a que el volumen de información que se debe procesar es grande, dicho algoritmo contempla la aplicación del modelo de agentes para controlar de manera dinámica el espacio de búsqueda y, en consecuencia, reducir los tiempos de procesamiento de los textos.

Palabras clave: *Text Mining, Attributed Relational Graphs, Agentes.*

Abstract

Text mining can be defined as the discovery of knowledge in very large collections of documents. It is associated to the discovery of interesting patterns like clusters, associations, deviations, similarities, and differences. On the other hand, the Attributed Relational Graphs (ARG) are defined as an extension of the ordinary graphs associating discrete or real attributes with their vertexes and arcs. The use of the attributes allows to the ARG to be possible of not only represent topological structures of an entity but also it's not structural properties, which usually can be represented as vectors. This characteristic makes ARG a useful technique for pattern detection. In this work, we define a pattern detection algorithm based on graphs. Due to the fact that the volume of information that it is necessary to process is big, the above mentioned algorithm contemplates the application of the agents to control, in a dynamic way, the space of search and, in consequence, to reduce the processing time.

***Key words**: Text Mining, Attributed Relational Graphs, Agents.*

Contenido

CAPITULO 1

Introducción

1.1. Definiciones Generales

El presente trabajo se fundamente sobre tres temas: Text Mining, Grafos y el concepto de agentes.

Text Mining es un área de investigación emergente que puede ser caracterizada como el descubrimiento de conocimiento en grandes colecciones de documentos, combinando métodos de aprendizaje con métodos de procesamiento de textos. Está asociado principalmente al descubrimiento de patrones interesantes como clusters, asociaciones, desviaciones, similitudes, y diferencias [10, 12, 29].

Asimismo, la búsqueda de patrones en textos es una herramienta útil en aplicaciones para reconocimiento inteligente de caracteres, sistemas de compresión de texto, traducciones automáticas, y aplicaciones similares en las que un sistema debe elegir el siguiente elemento (letra, palabra, fonema, etc...) de entre una lista de posibles candidatos [25].

Por otro lado, los Attributed Relational Graphs (ARG) se definen como una extensión de los grafos ordinarios asociando atributos discretos o reales a sus vértices y arcos. El uso de los atributos permite a los ARG ser posibles de no sólo modelar estructuras topológicas de una entidad sino también sus propiedades no estructurales, que usualmente se pueden representar como vectores [45]. Estas características hacen a esta herramienta un elemento útil a la hora de realizar búsqueda de patrones [39, 40, 44].

Finalmente, existe una gran variedad de definiciones de agentes dependiendo del uso que el autor le haya dado al término. Según la definición de agente de IBM, los agentes son entidades de software que ejecutan un conjunto de operaciones en nombre de un usuario u otro programa con cierto grado de independencia o autonomía, utilizando algún conocimiento o representación de las metas y deseos del usuario [14]. Por otro lado, investigadores de la Universidad de Indiana forman parte de una comunidad que apoya el enfoque basado en agentes cuyo conocimiento y capacidad deductiva sea limitado. Dicho enfoque propone que un conjunto de agentes simples permite la ejecución de un sistema inteligente de forma más sencilla [2]. Estas dos definiciones se fusionan en esta tesis con el fin de generar un conjunto de agentes que ejecuten una variedad de operaciones sencillas a partir de una representación de las metas y deseos del usuario. Este comportamiento se modeliza mediante la metodología BDI (Belief, Desire and Intentions) [20], que permite representar el accionar de un agente a partir del conocimiento que posea, las metas y sus intenciones.

1.2. Contribuciones

En el contexto presentado en la sección anterior, esta tesis define un modelo genérico basado en los ARG para la búsqueda de patrones con la incorporación de la teoría de agentes para la reducción de caminos irrelevantes en el grafo. Principalmente, se obtiene:

- Flexibilidad en el tipo de patrón que se desea detectar:

  El modelo presentado admite un conjunto de parámetros que permiten configurar la relación existente entre gramas de un texto en función de la distancia existente entre ellos. Dicha configuración se establece a partir de la definición de una función peso que pondera las distancias antes mencionadas.

- Control del volumen del grafo con la reducción de caminos irrelevantes.

  Mediante la incorporación de la teoría de agentes y en función de las creencias, los deseos y las intenciones de los agentes definidos en el modelo, se establece un control sobre el volumen del grafo seleccionando aquellos arcos más relevantes para el mismo.

- Velocidad en la detección de patrones.

  A partir de la incorporación de los agentes al modelo de grafos se reducen los tiempos de búsqueda de patrones.

Adicionalmente para este trabajo se desarrolló un software que permite, a partir de la búsqueda de patrones en textos, la categorización de los mismos y se estudió el impacto en memoria y tiempo de las incorporaciones mencionadas en esta tesis.

1.3. Trabajos Relacionados

Existen trabajos en la bibliografía actual que utilizan la representación de textos mediante grafos. Por ejemplo, en (Wei Jin, Rohini Srihari [18]) se propone un método basado en grafos para capturar la estructura y la semántica del documento de forma más efectiva. Básicamente el modelo se basa en un grafo que pondera las relaciones entre los elementos del documento que analiza y extrae conclusiones estructurales a partir de ellas.

En (Tomita,Nakawatase, Ishii [19]), el articulo se concentra principalmente en determinar los algoritmos para obtener subgrafos a partir del grafo original y calcular la similitud entre ellos. Por otro lado, en (Tomita,Nakawatase, Ishii [26]) se define una arquitectura para representación de paginas Web mediante grafos y su posterior almacenamiento para el descubrimiento de conocimiento en la Web.

Finalmente, existen publicaciones, como (Tsuyoshi Kitani[22]) que basan su trabajo en un idioma en particular (en el caso de [22], el Japonés).

La diferencia de este trabajo con los otros es que se presenta un modelo genérico para la búsqueda de patrones en textos que permite al usuario definir mediante una función peso el tipo de patrón que desea buscar y que realiza un control sobre el volumen del grafo para obtener mayor velocidad de procesamiento. Es decir, lo determinado en este artículo es independiente del objetivo que tenga el usuario, el idioma o las características del procesamiento que se le quiera dar al texto.

1.4. Área de aplicación

Existe un amplio rango de aplicaciones del Text Mining, y más particularmente, la búsqueda de patrones en textos. En primer lugar se puede mencionar la categorización de textos que se puede interpretar como la asignación a un texto de una o más categorías que estén relacionadas con su contenido. Por otro lado, el proyecto de la Web Semántica [27], que intenta esquematizar la Web a partir de tópicos de forma automática. En tercer lugar, la compresión de textos a partir del reconocimiento de patrones [29]. Y finalmente, el filtrado de mails que permite seleccionar los mismos de acuerdo a su relevancia.

1.5. Estructura de la Tesis

El presente documento se organiza de la siguiente manera: en el segundo capitulo se introduce el concepto de grafos y los elementos asociados a la teoría de grafos que serán de relevancia en el desarrollo de la tesis. En el capítulo tres se desarrollan los temas relacionados a la teoría de agentes desatacando conceptos como los dumb agents y la metodología BDI que son de importancia para el entendimiento del modelo presentado en este trabajo. En el cuarto capítulo se definen los conceptos vinculados al Text Mining y aquellos relacionados con el procesamiento de gramas. En el quinto y sexto capítulo se desarrolla el modelo basado en grafos que forma la base central de esta tesis. En el sexto capítulo, particularmente, se realizan aclaraciones vinculadas al modelo que se presenta en el quinto capitulo, las mismas son de relevancia para tener un entendimiento global del modelo. En el séptimo capítulo se muestran los resultados obtenidos mediante la corrida del software implementado y en el octavo capítulo se concluyen los aspectos más relevantes aportados por esta tesis.

CAPITULO 2

Grafos

Uno de los temas pilares en el presente trabajo es el estudio de los grafos. En este capitulo se pretende interiorizar en los conceptos más relevantes asociados a los grafos para poder comprender el objeto central de la tesis. No se intenta explicar toda la teoría de grafos sino aquella que será utilizada en este documento. Debido a que la terminología de la teoría de grafos no es estándar, el lector puede encontrar algunas diferencias entre los términos utilizados aquí y en otros textos. No obstante a lo largo del presente capitulo se intenta remarcar las principales diferencias encontradas.

En la primera sección, se introducen los conceptos más generales de la teoría de grafos que van a ser utilizados a lo largo del documento. Temas como, por ejemplo, la definición de grafos, arcos incidentes y grafo completo son analizados en esta primera parte.

La segunda y la tercera sección hacen hincapié en los grafos dirigidos y los grafos ponderados respectivamente. Se particularizan ambos temas debido a la estrecha relación que guardan con el grafo presentado como modelo para la tesis.

Por otro lado, la cuarta sección trata los temas de grado de un vértice y se determinan las definiciones más relevantes asociadas con este tema.

Finalmente, en la quinta sección se introducen los conceptos relacionados con caminos y ciclos de un grafo y se puntualizan detalles relacionados con los grafos dirigidos. Mientras que en la sexta sección se desarrollan nociones asociadas a los subgrafos y la obtención de los mismos.

2.1 Definición de grafo

Muchas situaciones y problemas de la vida real pueden ser representados mediante grafos. El uso de los mismos como modelos de escenarios complejos puede ser muy variado. Básicamente, todo esquema consistente en elementos que se relacionan entre sí puede ser modelado mediante grafos. Ejemplos de esto son las redes de telecomunicaciones, circuitos eléctricos, personas relacionadas en un ámbito laboral, y, como en el caso de este trabajo, gramas de un texto.

Formalmente un grafo se define como una tripla ordenada $[V(G), E(G), \psi_G]$, donde $V(G)$ representa al conjunto de vértices, $E(G)$ disjunto de $V(G)$, define al conjunto de aristas del grafo y la función de incidencia ψ_G que asocia cada arista de G con un par de vértices de G [5]. Si $e \in E(G)$ y $u, v \in V(G)$, tal que $\psi_G(e) = uv$, entonces se dice que e une a u y v, y estos son extremos de e. Básicamente, la función ψ_G ofrece una nomenclatura para representar las aristas del grafo.

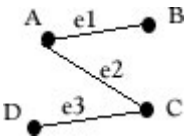


Fig. 2.1. Grafo

$$V = \{A, B, C, D\}$$
$$E = \{e_1, e_2, e_3\}$$
$$\psi_G(e_1) = AB; \psi_G(e_2) = AC; \psi_G(e_3) = DC$$

En lo referido a la cardinalidad de estos conjuntos $E(G)$ y $V(G)$, el número de vértices de un grafo se denomina orden, y se denota como $\nu(G)$ [1], asimismo, se utiliza $\varepsilon(G)$ para simbolizar el número de aristas. Los grafos pueden ser finitos, infinitos y numerables dependiendo del orden del mismo. Un grafo es finito solo sí tanto $\nu(G)$ como $\varepsilon(G)$ son finitos [5], en este documento siempre que se mencione el termino 'grafo' se referirá exclusivamente a los grafos finitos salvo que se mencione lo contrario.

Por otro lado, en el aspecto gráfico, un grafo puede ser representado como un conjunto de puntos unidos por líneas. Cada uno de los puntos implica un vértice, mientras que las líneas simbolizan las aristas. La topología del mismo así como las distintas características de los vértices y arcos, establecen las variadas clasificaciones del grafo. Dichas particularidades no forman parte del alcance de esta primera introducción, sino que serán atendidas en las secciones siguientes.

[1] El orden del grafo también puede ser representado como $\| G \|$ [41]

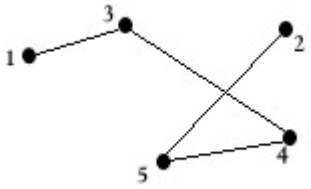


Fig. 2.2. Grafo $V = \{1,2,3,4,5\}$
$E(G) = \{\{1,3\};\{3,4\},\{5,2\};\{5,4\}\}$

Un vértice v es *incidente* con una arista e si $v \in e$. Los dos vértices incidentes con una arista se denominan extremos de la arista. De esta forma, si observamos la figura *2.1*, el vértice 3 es incidente con la arista $e_{34} = \{3,4\}$ y los vértices 3 y 4 son extremos de la arista e_{34}. Por otro lado, dos vértices x, y son *vecinos* o *adyacentes* si existe e la arista $e_{xy} = \{x, y\}$. En el caso de la figura *2.1* los vértices 1 y 3 son adyacentes. De la misma forma, dos aristas se denominan *adyacentes* si tienen un extremo en común, en nuestro ejemplo las aristas e_{13} y e_{34}. Si todos los vértices del grafo son adyacentes de a pares, entonces el grafo se denomina completo *[2]*. Es decir, Si $\neg\exists(a,b) \in E \rightarrow \exists(b,a) \in E$.

2.2 Grafos dirigidos

Supongamos que mediante un grafo se desean representar los puntos más importantes de la ciudad y las respectivas calles que los conectan. Como se puede prever los sitios de la ciudad serán vértices de nuestro grafo, mientras que las calles serán arcos. Es bien conocido que algunas calles pueden ser transitadas en ambos sentidos, mientras que otras no. Supongamos que todas las calles de nuestro ejemplo son del segundo tipo. De esta forma, si un punto A de la ciudad se conecta con un punto B mediante una calle y por ella solo se puede transitar en sentido AB, el arco e_{AB} de nuestro grafo se debe representar como el par ordenado $e_{AB} = \{A, B\}$, pero no al revés. Nuestro grafo es entonces un claro ejemplo de un grafo dirigido, o digrafo. Gráficamente las aristas de los grafos dirigidos se representan mediante flechas y se denominan arcos.

Formalizando este concepto, se define a un *grafo dirigido* D como una tripla ordenada $(V(D), A(D), \psi_D)$ que consiste en un conjunto $V(D)$ de vértices, un conjunto $A(D)$ disjunto de $V(D)$, de arcos, y una *función de incidencia* ψ_D que asocia a cada arco de D un par ordenado de vértices de D.

Existe una correlación entre los grafos dirigidos y los grafos, dado un grafo G cualquiera se puede obtener un digrafo de G especificando para cada vinculo de G una dirección. El digrafo resultante se denomina *orientación de* G. Por otro lado, todo concepto que sea valido para grafos es valido para los digrafos. No obstante, nuevas terminologías y simbologías se asocian exclusivamente a los grafos

dirigidos. En principio, como se mencionó anteriormente, un arco es un par ordenado de vértices, lo cual deriva en una nueva denotación para aquellos vértices conectados, supongamos un arco $a = \{u, v\}$, u de denomina *cola* de a, mientras que v es la *cabeza* de a [2]. En segundo lugar, dos nuevos conjuntos se tienen en consideración para el estudio de los digrafos, estos se están relacionados con los arcos salientes y los incidentes a un nodo. Formalmente, se simboliza $\bigcup_v^+$ al conjunto de arcos salientes de v, mientras que $\bigcup_v^-$ simboliza al conjunto de arcos incidentes a v [3]. La figura *2.2* ejemplifica lo mencionado anteriormente.

Otra definición relevante para el estudio de digrafos es el *grafo no dirigido asociado*. Simplemente, éste consiste en el reemplazo de los arcos del digrafo por aristas (par de vértices no ordenado). Si se obtiene más de una arista no dirigida de un par de vértices distintos de, entonces sólo una de estas aristas se dibuja en el grafo no dirigido asociado.

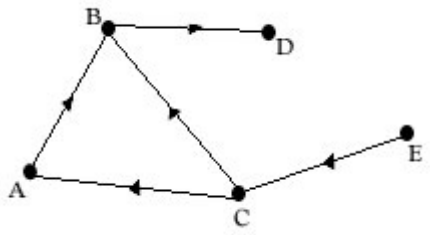


Fig. 2.3. Digrafo
$\bigcup_B^+ = \{\{B, D\}\}$
$\bigcup_B^- = \{\{A, B\}, \{C, B\}\}$

En esta sección solo se pretende introducir el concepto de digrafo, es por ello que no se desarrollan todos los puntos referidos a este tema, sin embargo, a lo largo del presente documento se apuntarán particularidades referidas a los digrafos en los diversos ítems abordados.

2.3. Grafos Ponderados

Siguiendo con el ejemplo de la sección anterior, supongamos ahora que deseamos viajar de un punto de la ciudad a otro pero teniendo en cuenta el costo del traslado. Dejemos de lado el hecho de la imposibilidad de transitar las calles en ambos sentidos y solo focalicémonos en el costo. De esta manera, a cada arco del grafo le corresponde un monto referido al traslado de un sitio a otro de la ciudad, es decir una ponderación. Este grafo recibe la denominación de *grafo ponderado.*

Formalmente, sea un grafo $G = (V, E)$ y existe, por otro lado, una función cualquiera del tipo $w : E \to \Re$ llamada *función peso*. El grafo G, junto con la función w se denomina *grafo ponderado*.

[2] Términos obtenidos de [41]
[3] Simbología utilizada en [3]

Consideremos el siguiente ejemplo,

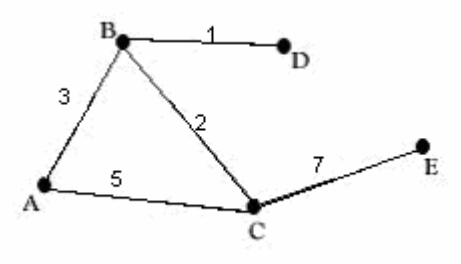


Fig. 2.4. Grafo ponderado

El grafo se define como $G = \{V, E)$ con

$$V = \{A, B, C, D, E\}$$
$$E = \{\{A, C\}, \{A, B\}, \{B, D\}, \{C, B\}, \{C, E\}\}$$

Y w se define por

$$w(\{A, B\}) = 3 \quad w(\{A, C\}) = 5$$
$$w(\{B, D\}) = 1 \quad w(\{C, B\}) = 2$$
$$w(\{C, E\}) = 7$$

Debe dejarse claro que si bien no fue considerado en esta sección, la posibilidad de un grafo dirigido y ponderado existe. La definición del mismo es análoga a la anterior: sea un grafo dirigido $G = (V, A)$ y existe, por otro lado, una función cualquiera del tipo $w : A \to N$ llamada *función peso*. El digrafo G, junto con la función w se denomina *digrafo ponderado*. En este caso, la *función peso* considera el par ordenado de vértices que determinan un arco, por lo que es posible que dos arcos con igual extremos posean una ponderación distinta. Es decir, supongamos un arco $a_{AB} = \{A, B\}$ y un arco $a_{BA} = \{B, A\}$, los dos poseen los mismos extremos, sin embargo, el orden del par es lo que varía, por lo que la función de ponderación puede devolver valores distintos.

Los problemas más relacionados con los grafos ponderados son los de las búsquedas del mejor camino. Los conceptos de camino y ciclo serán considerados en las secciones venideras.

2.4. Attributed Relational Graphs (ARG)

Los Atributted Relational Graphs fueron desarrollados para representar tanto información estructural como semántica de un objeto [38]. Los ARG se definen como una extensión de los grafos ordinarios asociando atributos discretos o reales a sus vértices y arcos. El uso de los atributos permite a los ARG ser

posibles de no sólo modelar estructuras topológicas de una entidad sino también sus propiedades no estructurales, que usualmente se pueden representar como vectores [45].

En los ARG el grafo en sí representa la estructura global del objeto, mientras que los vértices y las aristas determinan características locales del mismo, es decir las propiedades de los elementos que conforman el todo. Los ARG son utilizados para la descripción de un objeto que requiera ser definido a partir de las características de los elementos que lo conforman. Supongamos se desea modelizar una bicicleta con un grafo, cada parte de la bicicleta se encontraría definida mediante un nodo, por otro lado, la relación existente entre ellos, como por ejemplo la relación "esta unido a", es modelizada por las aristas. Cada vértice del grafo contiene diferentes propiedades, por ejemplo, las ruedas serán de un material determinado y de un diámetro definido. Sin embargo, la bicicleta en sí solo puede ser descripta a través de la descripción de sus componentes, esta es la funcionalidad que cubren los ARG.

Las aplicaciones de los ARG son muy diversas, como por ejemplo, reconocimiento de patrones sintácticos, análisis de escenarios, descripción de estructuras químicas, bases de datos relacionales, etc. [39, 40, 46].

2.5. Grado de un vértice

Supongamos que se tiene un grafo $G = (V, E)$, dos vértices $u, v \in V$ se dicen adyacentes o vecinos si existe una arista $e \in E$ tal que $u \in e$ y $v \in e$. De esta manera, se denota al conjunto de nodos[4] adyacentes a un vértice v como $N_G(v)$ y se llama grado o valencia del vértice v, $d(v)$, a la cardinalidad de dicho conjunto. Si todos los vértices tienen el mismo grado *k*, entonces, el grafo se denomina *k-regular* o simplemente *regular*. Como colorario se define a $\delta(G)$ y $\Delta(G)$ como la mínima y máxima valencia del grafo. Formalmente,

$$\delta(G) = \min\{d(v) \mid v \in V\}$$
$$\Delta(G) = \max\{d(v) \mid v \in V\}$$

Una medida interesante para el estudio de grafos es una aproximación al número de aristas por vértice, este valor se define como el grado promedio de G y se encuentra ligado a la siguiente expresión:

$$d(G) = \frac{1}{v(G)} \sum_{v \in V} d(v)$$

Debido a que si sumamos los grados de todos los vértices se obtiene dos veces el número de aristas. Es por ello que el número de aristas de un grafo G se puede calcular como:

[4] En el presente documento se utilizan los términos nodos y vértices como sinónimos.

$$\varepsilon(G) = \frac{1}{2}\sum_{v \in V} d(v) = \frac{1}{2} d(G).v(G)$$

Lo dicho hasta aquí es valido para cualquier grafo, no obstante, para el caso de grafos dirigidos se distinguen dos grados de vértices el grado entrante y el grado saliente (in degree y out degree). Básicamente, el grado entrante de un vértice representa el número de arcos incidentes al vértice, mientras que el grado saliente implica el número de arcos incidentes exterior del vértice. La simbología para representar estos conceptos es la utilizada por [34] $\deg_G^-(v)$ y $\deg_G^+(v)$. Lo expresado anteriormente puede resumirse en las siguientes expresiones:

$$\deg_G^-(v) = |\cup_v^-|$$
$$\deg_G^+(v) = |\cup_v^+|$$

De forma análoga para grafos dirigidos se define:

$$\delta^+(G) = \min\{\deg_G^+(v) \mid v \in V\},\ \delta^-(G) = \min\{\deg_G^-(v) \mid v \in V\}$$
$$\Delta^+(G) = \max\{\deg_G^+(v) \mid v \in V\}, \Delta^-(G) = \max\{\deg_G^-(v) \mid v \in V\}$$

Teorema: Si $G = (V, E)$ es un grafo dirigido con $\varepsilon(G)$ cantidad de arcos, entonces:

$$\sum_{v \in V} \deg_G^+(v) = \sum_{v \in V} \deg_G^-(v) = \varepsilon(G)$$

Prueba: sea $e_{uv} = \{u, v\} \in E$, es claro que:

$$\cup_u^+ \cap \cup_v^- = e_{uv}$$

Luego,

$$\bigcup_{v \in V}\left(\cup_v^+\right) = \bigcup_{v \in V}\left(\cup_v^-\right)$$

Por otro lado $\neg\exists e \in E / e \notin \bigcup_{v \in V}\left(\cup_v^+\right)$, entonces,

$$\bigcup_{v \in V}\left(\cup_v^+\right) = \bigcup_{v \in V}\left(\cup_v^-\right) = E$$

Finalmente,

$$\left|\bigcup_{v \in V}\left(\cup_v^+\right)\right| = \left|\bigcup_{v \in V}\left(\cup_v^-\right)\right| = |E|$$

$$\sum_{v \in V} \deg_G^+(v) = \sum_{v \in V} \deg_G^-(v) = \varepsilon(G)$$

2.6. Caminos y Circuitos

Volvamos por un momento al ejemplo presentado en la sección *2.2.* Queremos ahora, dejando de lado todas las limitaciones viales y monetarias, recorrer todos los puntos de la ciudad en un solo viaje. El mapa de nuestra ciudad es el siguiente grafo:

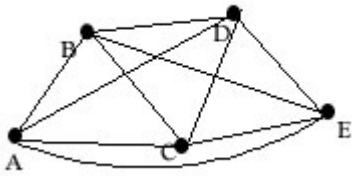


Fig. 2.5. Grafo Completo

Como se puede observar el grafo que se presenta es completo, es decir, para nuestro ejemplo, se puede viajar desde una ciudad a la otra sin problema. Supongamos que en nuestro afán de recorrer la ciudad partimos del punto A, pasando luego por B, D, C, y finalmente por E. De esta manera podemos afirmar que nuestro *camino* fue $\{A, B, D, C, E\}$. Sin embargo, si luego de transitar por todos los sitios del mapa, se decide retornar al punto de salida ya podemos decir que lo efectuado fue el *circuito* $\{A, B, D, C, E, A\}$. Es necesario destacar que ninguno de los dos conceptos esta asociado a el paso por todos los vértices del grafo, sino que los conjuntos $\{A, C, E\}$ y $\{A, C, E, A\}$, son también camino y circuito respectivamente. Una vez destacada la principal diferencia entre un camino y un circuito, se procede a la formalización de los conceptos:

Sea un grafo $G = (V, E)$, se llama *camino*, a una secuencia $\{e_1, e_2, ...\}$ de aristas tal, que la extremidad terminal de cada arista coincida con la extremidad terminal de la arista siguiente. Otra forma de representar un camino es a partir de los vértices que recorre. Un camino puede ser *compuesto* o *sencillo*, en el primer caso utiliza dos veces una misma arista, en el segundo no. Asimismo, se dice que un camino es *elemental* si no encuentra dos veces el mismo vértice. Por otro lado, un *circuito* es un camino finito $\mu = \{x_1, x_2, ..., x_k\}$, donde $x_1 = x_k$.

Dos caminos se denominan independientes si ninguno de ellos contiene un vértice interior del otro. De esta manera, un camino $\{A, C, D, E\}$ es independiente de $\{A, B, E\}$. Por otra parte, la longitud de un camino se encuentra referida a la cantidad de aristas o vértices que tenga. Dos vértices se encuentran conectados si existe un camino que los una.

Se dice que un grafo no dirigido es fuertemente conexo si para cualquier par de vértices (x, y) con $x \neq y$, existe un camino que va de x a y. Por otro lado, un grafo dirigido se dice conexo cuando su grafo no dirigido asociado es conexo.

Finalmente, todo lo expresado en esta sección cabe para los digrafos y los grafos ponderados, sin embargo, para los primeros es necesario destacar la irreflexibilidad de los caminos y ciclos. Algunos autores, como por ejemplo [3], hacen la distinción entre cadena y camino, esta diferencia radica principalmente en la misma discrepancia entre un arco y una arista. Básicamente, una cadena posee el mismo concepto que un camino, salvo que esta no se compone de arcos sino de aristas. En nuestro caso vamos a hablar de cadena o camino como sinónimos, entendiendo la existencia de arcos o aristas a partir de si el grafo que se utilice se declara como un grafo dirigido o no. Las nociones de camino, arco y circuito sirven para categorizar los digrafos. Se dice que un digrafo es *simétrico* si se tiene,

$$\text{Sea } G = (V, A) \text{ un digrafo, si } (x, y) \in A \Rightarrow (y, x) \in A$$

De la misma manera se dice que un digrafo es *antisimétrico* si

$$\text{Sea } G = (V, A) \text{ un digrafo, si } (x, y) \in A \Rightarrow (y, x) \notin A$$

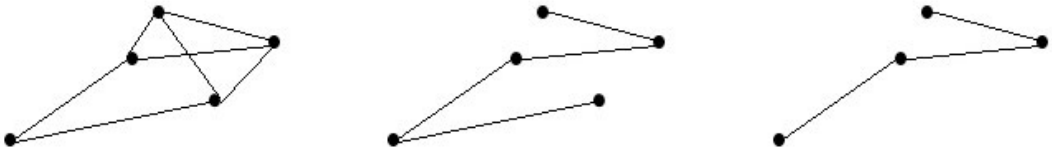
Por otro lado, para los grafos ponderados, la teoría de los caminos y ciclos se encuentra estrechamente relacionada a la búsqueda de aquel camino o ciclo cuya sumatoria de pesos sea mínima.

2.7. Subgrafos

Muchas veces para resolver un problema debemos partir de una estructura general para luego enfocarnos en una parte de esa estructura. Sigamos con el ejemplo de los grafos para representar mapas. Supongamos que tenemos un grafo que modeliza todas las calles de la Argentina, sin embargo, nuestro problema se encuentra referido a los puntos y calles más importantes de la Ciudad de Buenos Aires. Es por ello que debemos obtener lo que denominamos un *subgrafo* del grafo principal que represente a los elementos que necesitamos para nuestro objetivo.

Un grafo H es un subgrafo de G $(H \subseteq G)$ si $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ y ψ_H es la restricción de ψ_G para $E(H)$. Inversamente, si H es un subgrafo de G, entonces G es un *supergrafo* de H. Si $H \subset G$, entonces H se denomina *proper subgraph*. Si $V(H) = V(G)$, entonces H se denomina *spanning subgraph* de G.

Supongamos $V' \subseteq V$. El subgrafo de G cuyos vértices pertenecen a V' y cuyas aristas tienen ambos extremos en V' se denomina *subgrafo de* G *inducido por* V' y se denota como $G[V']$. De forma análoga se definen los *subgrafo arco-inducido de* G, como el subgrafo cuyas aristas pertenecen a un conjunto $E' \subseteq E$, y se denota como $G[E']$. En ambos casos, la construcción de un subgrafo *inducido* o un subgrafo *arco-inducido* corresponde a la eliminación de vértices o aristas, respectivamente. Es decir, para obtener uno de estos subgrafos no se debe agregar al grafo ni aristas ni vértices. En la figura *2.7.* se muestra un grafo y sus respectivos *subgrafo inducido y arco-inducido.*



A B C

Fig. 2.6. A) Grafo G. B) Subgrafo de G arco-inducido. C) Subgrafo de G inducido

Las operaciones de intersección y unión son realizables sobre los grafos. Supongamos se tienen dos subgrafos H_1 y H_2, se dice que son disjuntos si $V(H_1) \cap V(H_2) = 0$, y se dice *arco-disjuntos* si $E(H_1) \cap E(H_2) = 0$. Por otra parte, la *unión* $(H_1 \cup H_2)$ y la *intersección* $(H_1 \cap H_2)$, se efectúa realizando la unión y la intersección de los conjuntos de vértices y aristas de los subgrafos.

Finalmente, Sea un grafo $G = (V, E)$ y sea el conjunto de subgrafos fuertemente conexos de G, $Q = \{G_1' = (V_1', E_1'), G_2' = (V_2', E_2'), ..., G_k' = (V_k', E_K')\}$, con $\bigcap_{G_i' \in Q} G_i' = 0$, si se reemplaza cada elemento de Q por un nuevo vértice, entonces el grafo resultante se denomina *grafo reducido de* G. Esta definición también es extensible a grafos dirigidos. La siguiente figura ilustra este concepto:

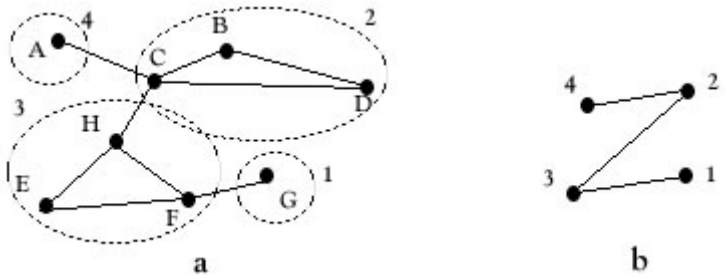


Fig 2.7. a) Grafo G, b) Grafo reducido de G

CAPITULO 3

Agentes

En este capítulo se introduce la teoría de agentes, la cual es otro de los temas pilares de este trabajo. Fundamentalmente, se pretende brindar el conocimiento básico necesario para el entendimiento de la modelización realizada en la resolución del problema de la búsqueda de patrones en textos.

En la primera sección se define el concepto general de agente, se lo diferencia con un programa convencional, se define el entorno de un agente y se introducen algunas clasificaciones del mismo.

En la segunda sección se aborda el tema de la clasificación de un agente en función del conocimiento que posea y su forma de operar.

En la tercera sección se define la estructura interna de un agente y la forma de comparación de la misma con la de otros agentes, mientras que en la cuarta sección se introducen los conceptos sobre los sistemas multiagente y los dos enfoques existentes para construirlos.

En la quinta sección se definen las metodología basada en POO y la metodología basada en ICO para el desarrollo de sistemas agentes.

En la sexta sección se define uno de los temas pilares para esta tesis que es el concepto de "dumb agent". Finalmente se definen las características del modelo BDI para la modelización del estado del agente como así también su comportamiento.

3.1 Definición de agente

Son muchas las acepciones que tiene el término agente, y dependen del autor que las pronuncie. Sin embargo, todas las definiciones tienen en cuenta dos ideas fundamentales [14]:

1. Es algo que actúa, o puede actuar.
2. Alguien que actúa en lugar de otro, con permiso, información y un propósito.

Si bien estos dos conceptos pueden servir como pautas para determinar una enunciación más formal sobre el término agente, tal vez unos ejemplos puedan servir para aclarar el panorama y poder concluir en una frase significativa.

Como ejemplos de agentes se puede considerar cualquier humano o animal, como también los robots autónomos o agentes de software que "viven" en sistemas operativos, bases de datos, redes, etc. Entonces, ¿qué comparten todos estos agentes que constituye la esencia de ser un agente?

En principio, todos forman parte de un ambiente del cual obtienen la información, que les permite actuar de forma autónoma. Por otro lado, todos, en mayor o menor medida, tienen una meta que alcanzar y sus acciones repercuten en el medio. Por lo tanto, es concebible formalizar lo siguiente [14]:

> *Los agentes son sistemas autónomos que intercambian información con el ambiente en el que están subscriptos para alcanzar un objetivo.*

Como se puede observar la definición de un agente es bastante amplia y, fácilmente, se la puede comparar y confundir con la definición de un programa. Sin embargo existen diversas diferencias entre un programa y un agente. Como por ejemplo [14]:

1. Los agentes son autónomos. Esta diferencia se basa fundamentalmente en que un agente no necesita del accionar del usuario u otro sistema. Es decir, opera independiente de los usuarios.
2. Contienen cierto grado de inteligencia. El concepto de autonomía explicado anteriormente implica que el agente posea inteligencia para la toma de decisiones. Esta capacidad se puede llevar a cabo mediante los llamados motores de inferencia que poseen reglas para deducir ciertos accionares considerando la información disponible.
3. No actúan de forma reactiva, pero si, a veces, proactiva.
4. Tienen la habilidad de trabajar socialmente con otros agentes para alcanzar un objetivo en común mucho más complejo. Se manejan diversos protocolos de comunicación para que un agente entienda a otro y pueda compartir los datos que posee.

Ahora bien, ya que determinamos las diferencias de un agente con un programa ordinario, si nos referimos a los agentes artificiales (software, robots, etc.), ¿cuáles son los beneficios que éstos otorgan?,

es decir, ¿Cuál sería un buen motivo para implementar un agente? La respuesta a estas preguntas se puede contestar mediante dos simples objetivos:

- Simplificar el manejo de información y procesamiento distribuido (Agente como administrador de recurso).
- Sobrellevar el problema de las interfaces de usuario (Agentes como asistentes personales que se adaptan al usuario).

Finalmente, para poder terminar de entender la definición de un agente y su forma de operar es necesario desarrollar un poco más el concepto de ambiente. Éste puede ser considerado como el medio donde esta sumergido el sistema, como, por ejemplo, la sección de producción de una fábrica, una controladora de una línea de producción, otro sistema, etc. El ambiente estimula al agente y le provee información para poder operar y tomar decisiones. Como el funcionar de un agente depende del ambiente es imprescindible establecer una clasificación [32]:

- Accesibles Vs. Inaccesibles: Un sistema accesible es aquel en el cual se puede obtener información completa, certera y actualizada sobre el estado del ambiente
- Determinísticos Vs. No-determinísticos: Un sistema es determinístico cuando ante la realización de una acción determinada la respuesta a la misma es siempre la misma.
- Episódicos Vs. No-episódicos: Un ambiente episódico es aquel en el cual la performance de un agente es dependiente de un discreto número de episodios sin enlace con el accionar del agente en diferentes escenarios.
- Estáticos Vs. Dinámicos: Un ambiente estático es aquel que permanece en el mismo estado a no ser que sea modificado por el accionar del agente.
- Discreto Vs. Continuos: Un ambiente es discreto si existen un número finito y cerrado de estados y acciones posibles sobre el mismo.

3.2. Clasificación de Agentes

Los agentes poseen diversas propiedades según el propósito que deban alcanzar. Lo cual deriva en la siguiente clasificación [14]

Propiedad	Significado
Reactivo	Responde inmediatamente al cambios en el ambiente
Autónomo	Tiene control sobre sus acciones.
Orientado a metas	No actúa en respuesta al ambiente, sino que sigue un plan para alcanzar su propósito.
Temporalmente continuo	Es un proceso que esta continuamente corriendo

Comunicativo, sociable	Tiene la capacidad de comunicarse con otros agentes, tal vez también con personas.
Adaptativo	Cambia su comportamiento de acuerdo a sus experiencias.
Móvil	Se puede transportar autónomamente hacia otro lado
Flexible	Sus acciones no están determinadas de ante mano
Personaje	Con personalidad y "estado emocional".

De acuerdo a nuestra definición un agente satisface las cuatro primeras características. Cualquier agregado de propiedades deriva en una subclase de agente.

Cabe mencionar que la categorización brindada anteriormente no es única, se puede clasificar a un agente de acuerdo a las metas que posea, a su arquitectura de control o al rango y sensibilidad de su percepción del medio.

3.3 Estructura interna de un agente

Si bien los agentes tienen características similares, éstos pueden diferir en su complejidad y los algoritmos que implementen internamente. Las diferencias se pueden observar en los siguientes dominios [30]*:*

- *Parecido (similarity):* Los agentes que no se comunican directamente entre sí, es decir lo que simplemente interactúan con el medio, pueden diferir completamente en su estructura interna. Sin embargo, si se comunican directamente uno con otro, deben hablar el mismo lenguaje. A veces los agentes comparten el mismo código y solo difieren en los parámetros de estado. Sin embargo, lo más usual es que comparten los encabezados pero el código interno es distinto. Se puede identificar tres tipos: diferentes, idénticos y los *body-head agents.* Éste último representa el caso en que los encabezados de los agentes son iguales pero su cuerpo no.

- *Modularidad (modularity):* El modelo que distingue los encabezados de los cuerpos de los agentes (body-head) permite la reutilización de los códigos de comunicación entre los agentes que son disímiles.

- *Memoria (memory):* Los agentes pueden o no recordar los cambios que sufrieron en su estado basándose en su experiencia.

- *Mutabilidad (mutability)*: En algunos sistemas, el código del agente puede cambiar a lo largo de su vida. Nos referimos fundamentalmente a la estructura de datos que esta manipulando en un momento determinado. Esta modificación puede ser impulsada desde fuera del agente o internamente.

3.4 Sistemas Multiagentes

Un agente racional es una entidad que hace lo correcto para cumplir con sus metas. Los Sistemas Multiagentes (SMA) buscan lograr la cooperación de un conjunto de agentes autónomos para la realización de una tarea. La cooperación depende de las interacciones entre los agentes e incorpora tres elementos: la colaboración, la coordinación y la resolución de conflictos.

Los SMA tienen un amplio rango de aplicaciones, uno de los campos principales la constituyen los agentes software los cuales se focalizan principalmente en la explotación cooperativa de recursos accesibles por Internet. El otro campo de aplicaciones incluye la utilización de agentes físicos y ha dado lugar a la robótica cooperativa; campo en el cual los conceptos y las teorías de SMA encuentran un mayor reto de aplicación. Algunos de los campos de aplicación de los SMA incluyen: asistentes personales, supervisión hospitalaria, asistentes financieros, manipulación y filtrado de información aplicados a diferentes dominios, agentes bancarios, ventas y comercio electrónico, difusión de noticias y publicidad, realidad virtual y avatares, control de procesos y manufactura, telecomunicaciones, entre otros.

Existen dos enfoques para construir sistemas multiagentes:

- El enfoque formal o clásico, que consiste en dotar de los agentes de la mayor inteligencia posible utilizando descripciones formales del problema a resolver y de hacer reposar el funcionamiento del sistema en tales capacidades cognitivas. Usualmente la inteligencia es definida utilizando un sistema formal (por ejemplo, sistemas de inferencia lógica) para la descripción, raciocinio, inferencia de nuevo conocimiento y planificación de acciones a realizar en el medio ambiente.
- El enfoque constructivista, que persigue la idea de brindarle inteligencia al conjunto de todos los agentes, para que a través de mecanismos ingeniosamente elaborados de interacción, el sistema mismo genere comportamiento inteligente que no necesariamente estaba planeado desde un principio o definido dentro de los agentes mismos (que pueden ser realmente simples). Este tipo de conducta es habitualmente llamado comportamiento emergente.

3.5 Metodologías de modelado de agentes

En los últimos tiempos, el crecimiento en investigación sobre sistemas Multiagentes se hace notorio, sin embargo, esta nueva tecnología es relativamente nueva, por lo tanto, las metodologías utilizadas se basan en metodologías existentes a las cuales se han modificado para adaptarlas al problema de los agentes.

3.5.1 Metodología OO

Existen varios beneficios que justifican las extensiones de la metodología orientada a objetos para la aplicación en sistemas Multiagentes.

En principio, existen notables similitudes entre el paradigma orientado a objetos y el paradigma de agentes: [13]

1. Paso de mensajes para comunicarse.
2. El empleo de herencia y agregación para definir su arquitectura.
3. La utilización de una vista estática para definir la estructura de los objetos y sus relaciones puede ser utilizada también con agentes.
4. Una vista dinámica para describir interacciones.
5. Una vista funcional para describir el flujo de datos entre los objetos, puede ser utilizado, para el caso de los agentes para hacer referencia a la comunicación de los mismo.

Por otro lado, existen diferencias significativas también puntualizadas por [13]*:*

1. Los agentes se comunican de una forma predeterminada
2. El estado mental de los agentes se determina (para una metodología BDI, ver sección 3.7) mediante creencias, intenciones, deseo y acuerdos.
3. Los agentes no son objetos por lo que las herramientas de modelado del paradigma orientado a objetos no explican todo lo relacionado con los agentes.
4. Los objetos no actúan de forma autónoma y no tienen la capacidad de inferir comportamiento como los agentes, en consecuencia, el modelado de agentes mediante técnicas de la orientación a objetos no esquematizan estas características.
5. La comunicación entre agentes determina un proceso más complejo, definido por un protocolo y por un pasaje de parámetros que se conocen de antemano. En el caso de los objetos, por otro lado, su comunicación se basa principalmente en la invocación de métodos por lo que lo hace mucho más sencillo en comparación con la de los agentes.

3.5.2 Metodología ICO

La idea fundamentalmente es que se utilicen todos los avances realizados en la ingeniería en el conocimiento en los Multiagentes para darle la característica de un objeto actuante y autónomo.

Dado que los agentes tienen características cognitivas, estas metodologías pueden proporcionar las técnicas de modelado de la base de conocimiento de los agentes.

La definición del conocimiento de un agente puede considerarse un *proceso de adquisición de conocimiento*, y dicho proceso sólo es abordado por éstas metodologías.

La extensión de metodologías de conocimiento puede aprovechar la experiencia adquirida en dichas metodologías. Además se pueden reutilizar las bibliotecas de métodos de resolución de problemas y ontologías así como las herramientas desarrolladas en estas metodologías.

Aunque estas metodologías han sido empleadas en ámbitos más restringidos que las orientadas a objetos, también han sido aplicadas con éxito en la industria.

La mayor parte de los problemas planteados en las metodologías de ingeniería del conocimiento también se dan, obviamente, en el diseño de sistemas multiagente: adquisición del conocimiento, modelado del conocimiento, representación y reutilización. Sin embargo, estas metodologías conciben un sistema basado en conocimiento como un sistema centralizado. Por tanto, no abordan los aspectos distribuidos o sociales de los agentes, ni en general su conducta proactiva, dirigida por objetivos.

3.6 Dumb Agent

Para la construcción de sistemas de agentes existe un enfoque basado en lo que denominamos un dumb agent (agente tonto). Fundamentalmente, esta idea propone la formulación de sistemas basados en agentes que realicen tareas simples y que no recurran a métodos de inteligencia artificial, sino que dicha inteligencia se genere a partir del accionar del conjunto de dumb agents.

La aplicación de dichos agentes puede presentarse tanto como agentes aislados intentando optimizar su rendimiento, como, en otros dominios, agentes que puedan interactuar, incluyendo a las actividades del mismo, la coordinación con el resto. Asimismo, el comportamiento de cada agente en el sistema puede ser el mismo o distinto, el sistema se evalúa de acuerdo al cumplimiento del objetivo común.

En [2] se presentan algunos sistemas desarrollados mediante esta metodología en la universidad de Indiana, como ser el movimiento de un robot o la aplicación a sistemas de comunicaciones. En todos ellos, se utiliza la idea de realizar actividades complejas a partir de un conjunto de procedimientos simples que son ejecutados por agentes que carecen de inteligencia o memoria.

Por lo general, los dumb agentes no tienen memoria, ni son capaces de aprender, sin embargo, son útiles al momento de desarrollar sistemas que impliquen un gran número de agentes y donde la toma de decisiones no es el principal objetivo. En estos casos, los dumb agents resultan beneficiosos en tiempo y uso de memoria.

Estos agentes simples, pueden ser clasificados de diversas maneras, si nos remitimos a la sección *3.2*, un dumb agent puede ser *reactivo* y *comunicativo*, como así también *orientado a metas* o comunicativo. Es decir, los agentes tontos no deben ser vistos como un simple procedimiento que ejecute tareas sino que puede ser implementado de diversas maneras dependiendo del objetivo que se desee alcanzar.

En resumen, el enfoque de desarrollo de sistemas Multiagentes basado en Dumb Agents se fundamenta principalmente en el uso de agentes simples que recurran a un escaso uso de los recursos de memoria para generar un sistema que realice una actividad compleja. Es decir, para ciertos problemas, como los presentados en [16], resulta más útil la implementación de sistemas con módulos operativos simples que la utilización de un conjunto de agentes complejos basados en las diversas técnicas de la inteligencia artificial.

3.7 Creencias, Deseos e Intenciones (BDI)

Si bien se presentó en las secciones anteriores algunas de las metodologías utilizadas para la modelización de los agentes, existen modelos que pretenden sentar los principios básicos para esquematizar los estados internos de un agente.

El modelo de Creencias, Deseos e Intenciones se ha convertido, posiblemente, en uno de los métodos más estudiados en lo que concierne a la teoría de agentes [16]. Estos tres conceptos fueron introducidos por [6] para explicar el funcionamiento del razonamiento humano. Estas ideas se formalizaron luego como Creencias, Metas y Planes, que implican una formalización más fácil de implementar. Sin embargo, la metodología se conoce aún como BDI (Belief, desires and intentions).

En principio, las Creencias de un Agente se encuentran relacionadas al entendimiento que tiene el mismo sobre el ambiente en el que esta inmerso. Es decir, representa de qué forma el Agente observa el medio y como lo entiende, cuál es la información que percibe y como la modeliza [37]. El modelo de creencias especifica la información que posee el agente y las acciones que puede realizar. Ese conjunto se denomina *conjunto de creencias* y puede variar o no según el estado del ambiente. Formalmente, el conjunto de creencias se define como un set de predicados cuyos argumentos son definidos por el usuario [21]. Si seguimos una metodología de modelado de agentes mediante objetos, este conjunto de predicados depende exclusivamente de la clase de agente que se esté implementando.

Por otro lado, las Metas representan el deseo y el objetivo del agente, es decir, lo que quiere cumplir.

Finalmente, los Planes implican las acciones que realizará el agente en función de la Meta que quiere lograr y en cuanto a los conocimientos o Creencias que percibe del medio [37]. A igual que para el caso de las creencias, los agentes poseen un conjunto de planes que describe las propiedades y las estructuras de control de los diversos planes [21]. La forma de esquematizar los planes individuales de cada uno de los agentes se puede plasmar como un pseudo código [33] o mediante un diagrama de estados [21]. El nivel de complejidad que tenga el agente para determinar cada uno de los Planes como así también su forma de entender y ambiente determinan las características y, en consecuencia, la clasificación del mismo.

Existen dos niveles de abstracción para los sistemas conformados por agentes [33]. En primer lugar, un punto de vista externo, que pretende esquematizar al sistema como una jerarquía de herencia de clases de agentes. Dichas clases se encuentran identificadas con las metas y planes del agente, como así también por la información que necesita del ambiente. Por otro lado, el punto de vista interno impone una estructura sobre la información que maneja el agente y la forma en que desarrolla sus planes.

CAPITULO 4

Text Mining

En este capitulo se aborda el último de los temas pilares de esta tesis. Principalmente se introduce el concepto de Text Mining y las operaciones posibles con palabras de un texto.

En la primera sección se define el concepto de Text Mining ejemplificando los usos de las distintas herramientas para el procesamiento de textos.

En la segunda sección se hace referencia a las definiciones básicas que serán utilizadas a lo largo del documento asociadas al procesamiento de textos y las palabras. Este conjunto de términos es de suma importancia para el entendimiento de las definiciones que se expliciten en los capítulos siguientes.

En la tercera sección se definen las operaciones con palabras que representan mayor relevancia para este trabajo como, por ejemplo, la concatenación y potencia.

En la cuarta sección se introduce el concepto de Asociación de palabras para luego, en las secciones cinco y seis, explicitar el soporte y confianza de una regla de asociación y su representación gráfica, respectivamente.

4.1. Introducción

La información existente en, por ejemplo, una empresa, puede encontrarse de forma estructurada, como en el caso de las base de datos, o destructurada como grandes colecciones de textos. Con respecto a la primera forma, el estudio de las bases de datos es desarrollado mediante técnicas de Data Mining (DM), mientras que la exploración de textos para la búsqueda de información concierne al área del Text Mining (TM). La importancia de la exploración de información radica en el soporte para la toma de decisiones, la automatización de actividades, entendimiento de desarrollos de procesos.

En la actualidad una gran porción de la información se encuentra en forma textual, y por lo tanto destructurada [31]. El hecho de no presentar una estructura definida hace que los textos propongan un desafío más complejo a la hora de su procesamiento. Es por ello que es necesario el desarrollo de técnicas especializadas que permitan extraer información relevante de grandes colecciones de documentos. Muchas de las aplicaciones de Text Mining pueden implicar el uso de herramientas basadas en sistemas de Procesamiento del Lenguaje Natural (NLP), utilizadas, por ejemplo, para el prepocesamiento de un texto o análisis semánticos [31].

Por otro lado, el análisis y la organización de grandes repositorios de textos son también de sumo interés en los ámbitos relacionados con la Inteligencia Artificial y el aprendizaje automático [23]. Fundamentalmente, el objetivo es incluir "algoritmos inteligentes" en los procesos de exploración de textos para automatizar la extracción de información relevante de los mismos. Ejemplos de la interconexión de dichos algoritmos puede observarse en trabajos como [1], [35] y [41].

Los desarrollos informáticos que impliquen la utilización de técnicas de Text Mining pueden ser muy variados, por ejemplo:

- Filtrado de Mails [23]: El usuario desea definir, a partir del contenido de sus mails, un filtro automático. El sistema posee dos fases, en la de entrenamiento, el algoritmo aprende a partir de los textos y las decisiones de filtrado por parte del usuario (aprendizaje supervisado) . Luego, el sistema clasifica los mails según lo determinado en la primera etapa y corrige sus aciertos o errores a partir de la supervisión del usuario.
- Clasificación de textos [23], [15], [35]: A partir de patrones o estructuras comunes en textos categorizar los mismos de forma automática. Esta aplicación del Text Mining se encuentra fuertemente ligada a la categorización de páginas Web mediante la cual se pueden realizar búsquedas en Internet a partir de conceptos y no palabras o, por otro lado, definir perfiles de usuarios.
- Lingüística computacional y procesamiento de textos [36]: Estudio del lenguaje a partir de métodos computacionales con un objetivo principal, la comprensión del lenguaje.

Según lo explicitado anteriormente se puede concluir la siguiente definición para el concepto de Text Mining:

"*Text Mining es un área de investigación emergente que puede ser caracterizada como el descubrimiento de conocimiento en grandes colecciones de documentos, combinando*

métodos de aprendizaje con métodos de procesamiento de textos. Está asociado principalmente al descubrimiento de patrones interesantes como clusters, asociaciones, desviaciones, similitudes, y diferencias."

Finalmente, en el aspecto práctico, cuando se habla de información contenida en documentos de textos, esta suele ser definida como relaciones o patrones que están inmersos en los documentos y que, sin las herramientas vinculadas al Text Mining, serían imposibles de identificar.

4.2. Definiciones básicas

Antes de introducirnos en los conceptos más significativos asociados al procesamiento de textos y Text Mining, definamos algunos de los ítems más simples pero relevantes para la modelización de un documento al momento de su estudio. Para referirnos a los temas enunciados en esta sección se utilizará la terminología y nomenclatura definida en [42].

4.2. a. Alfabeto

Un alfabeto $(\sum)$ puede definirse simplemente como un *conjunto no vacío finito de símbolos* [42].

Los símbolos pueden ser de cualquier tipo, como por ejemplo, letras, números, combinaciones de letras y números, etc. Es decir, un alfabeto debe ser definido a priori en cualquier análisis de textos y se puede especificar a partir de un conjunto de símbolos cualesquiera, según el tipo de procesamiento que se desea llevar a cabo.

Ejemplos de alfabetos pueden ser: el alfabeto español, el de los números o el alfabeto formado por los símbolos $\sum_a = \{a, si, solo, \Omega, \in\}$.

4.2. b. Palabra o Cadena

Una palabra o Cadena se define como una *secuencia finita de símbolos de un alfabeto*. [42]

Si tomamos por ejemplo, los alfabetos antes mencionados, en el caso del alfabeto español todos los términos expresados en este documento serían Palabras o Cadenas de ese alfabeto, por otro lado, el número "125" representaría una Cadena del alfabeto de los números, y "$\Omega si \in$" una palabra del alfabeto $\sum_a$.

Cabe destacar que se pueden definir palabras vacías (λ) como aquellas que no poseen ningún símbolo.

Las palabras o cadenas reciben también una denominación distinta de acuerdo al número de símbolos que posean (sección 4.2 c.). Una palabra de *k* símbolos se denomina *k-grama* [9], por ejemplo, "123" es una *tri-grama* y "ab" un *di-grama*. Asimismo en [9] se acuña el término de *multigramas* como un conjunto de gramas de longitud arbitraria.

Todas estas terminologías serán utilizadas a lo largo de este documento como sinónimos.

4.2. c. Longitud de una palabra

Para una palabra (α) de un alfabeto cualquiera, se define su longitud $|\alpha|$ como el *número de símbolos de alfabeto que la forman* [42].

Por ejemplo, si se toma la cadena "prueba", $|prueba| = 6$.

Es necesario mencionar, aunque trivialmente, que $|\lambda| = 0.$

4.2. d. Universo de un alfabeto

El universo de un alfabeto $W(\sum)$ corresponde al conjunto de todas las palabras que se pueden formar con los símbolos del alfabeto $\sum$ [42]. El número de elementos de $W(\sum)$ es infinito.

Si definimos el alfabeto $\sum_q = \{a, b\}$, $W(\sum_q) = \{a, b, ab, aa, bb, ba, bab, ...\}$

La palabra vacía pertenece a todos los universos.

4.2. e. Lenguaje sobre un alfabeto

Sea un alfabeto $\sum$, se dice que el conjunto finito $L(\sum)$ es un lenguaje sobre $\sum$ si todo elemento de $L(\sum)$ pertenece a $W(\sum)$. Debido a que $W(\sum)$ es infinito, existen infinidad de lenguajes sobre un mismo alfabeto.

Para el caso del alfabeto definido en *4.2. d.* se tiene:

$$L^1(\sum_q) = \{a, bb, aa, ba\}$$

$$L^2(\sum_q) = \{\lambda, b, bbb, bab\}$$

4.3. Operaciones con palabras

Otra de las cuestiones que se deben tener en cuenta al momento de procesar palabras es la posibilidad que se tiene de operar con ellas. Las operaciones posibles son la *concatenación*, *Potencia* y *Reflexión* [42]. Las mismas serán utilizadas, en parte, a lo largo del desarrollo del modelo (*ver capitulo 5*).

4.3. a. Concatenación

Supongamos, se tienen dos gramas α_1 y α_2, la concatenación de los mismos $\alpha_1.\alpha_2$, o simplemente $\alpha_1\alpha_2$, es una palabra formada por los símbolos de α_1 seguidos de los símbolos de α_2 [8].

Por ejemplo, supongamos el alfabeto $\sum = \{ante, bajo, contra, por\}$, se $\alpha_1 = ante \in \sum$ y $\alpha_2 = por \in \sum$, entonces $\alpha_1.\alpha_2 = antepor$. Es claro que para cualquier alfabeto $\sum, \alpha_1\alpha_2 \in W(\sum)$

4.3. b. Potencia

Se define a la potencia i-ésima de un grama α, cuando la misma se forma a partir de la concatenación de la palabra α *i* veces [42].

$$\alpha^i = \underbrace{\alpha.\alpha.\ldots\ldots\alpha}_{i}$$

Si tomamos, por ejemplo, el grama $\alpha = 20$, entonces $\alpha^3 = 202020$

4.3. c. Reflexión

Supongamos se tiene una palabra α conformada por los símbolos $A_1, A_2, ..., A_n$, entonces la palabra inversa de α, α^{-1}, se forma invirtiendo el orden de sus símbolos [42], es decir,

$$\alpha^{-1} = A_n.A_{n-1}.....A_1$$

De esta forma, si se tiene, por ejemplo, la palabra $\alpha = ropa$, entonces $\alpha^{-1} = apor$.

4.4. Asociaciones

Uno de los conceptos que se mencionaron en la sección 4.1 es la relación existente entre dos o más elementos de un texto. Estas relaciones sirven para definir características de un documento y a partir de ellas inducir algún tipo de conocimiento dependiendo del objetivo a cumplir. Para representar esta relación de palabras se utilizan las asociaciones.

Consideremos, en principio, un conjunto de palabras $\mathrm{A} = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ y un conjunto de Textos $T = \{t_1, t_2, ..., t_n\}$ y cada uno de los textos pertenecientes a T se encuentra asociado a un conjunto de elementos de A representado mediante $t_i(\mathrm{A})$.

Si tomamos un subconjunto de A denominado W, se define la cobertura de W como el conjunto de todos los textos pertenecientes a T tal que $W \subseteq t(\mathrm{A})$ y se simboliza como $[W]$ [31]. Es decir, todos los documentos cuyo conjunto de palabras asociado contenga en parte al conjunto W.

Ejemplo 4.1

Supongamos los siguientes conjuntos:

$$\mathrm{A} = \{10,00,11,01,000,001,100\}$$
$$T = \{t_1, t_2, t_3\}$$
$$t_1(\mathrm{A}) = \{10,000,001,11\}$$
$$t_2(\mathrm{A}) = \{000,11,01\}$$
$$t_3(\mathrm{A}) = \{10,000,001,11,01\}$$
$$W = \{10,000,001,11\}$$

entonces,

$$[W] = \{t_1, t_2\}$$

Finalmente, se denomina *regla de asociación* a todo par (W, α), donde $W \subseteq A$ y $\alpha \in A \backslash W$ y se denota como [31]:

$$R : W \rightarrow \alpha$$

Como se puede observar las reglas de asociación determinan la relación existente entre un conjunto de gramas y una palabra pertenecientes a un mismo conjunto A. Debido a que cada elemento del W se asocia a un documento en T, una asociación simboliza, indirectamente, la relación existente entre un conjunto de palabras y un texto.

4.5. Soporte y Confianza

Existen dos medidas que cuantifican la calidad de una determinada regla: el soporte y la confianza. Estos dos valores son también considerados como estimadores de la probabilidad condicional y de junta que estiman la fuerza de la asociación.

Otros autores definen de forma análoga, en lugar de soporte y confianza, los términos *prevalecencia* y *predictibilidad*, respectivamente. Sin embargo, en este documento se adopta la convención utilizada en [31] y [44].

4.5. a. Soporte

El soporte de una regla $R: W \rightarrow \alpha, W \subseteq \mathrm{A}, \alpha \in W \setminus \mathrm{A}$ se define como el porcentaje de ítems de A que satisfacen la unión de ítems en W y α [44, 31]. Esta definición se puede expresar matemáticamente de la siguiente manera:

$$S(R,T) = \left| \left[W \cup \{\alpha\} \right] \right|$$

Donde T es un conjunto de textos.

4.5. b. Confianza

La confianza es un estimador del porcentaje de documentos que cumplen con W y α [45, 32]. Formalmente, la confianza de R respecto al conjunto de textos T, se expresa como:

$$C(R,T) = \frac{\left| \left[W \cup \{\alpha\} \right] \right|}{\left| W \right|}$$

Observar que la $C(R,T)$ es una aproximación de la probabilidad de que un texto se asocie a la palabra α si ya está asociado a un término de W.

4.6. Visualización de una regla de asociación

Existen dos formas de visualizar las reglas de asociación producto de un trabajo de Text Mining, mediante una matriz bidimensional o un grafo dirigido [44]

4.6. a. Matriz Bidimensional

La matriz bidimensional es un método muy sencillo para representar una regla de asociación. Toda regla $R: W \rightarrow \alpha$ posee un antecedente (W) y un consecuente (α). Es decir, un lado izquierdo y un lado derecho de la regla. El método de la matriz bidimensional consiste en esquematizar a partir de una matriz una regla de asociación donde los ejes de la misma representan el antecedente y el consecuente de la regla. La conexión entre un antecedente y un consecuente determinan la regla y en esa conexión se representa gráficamente la confianza y el soporte de la misma.

Por ejemplo, si tomamos la regla:

$$R_1 : \alpha_1 \rightarrow \alpha_2$$

Tomando en consideración el alfabeto:

$$\sum = \{\alpha_1, \alpha_2, \alpha_3\}$$

La matriz que representa la regla R se dibujaría de la siguiente manera:

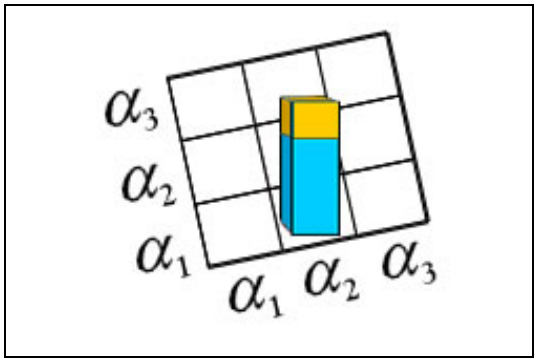
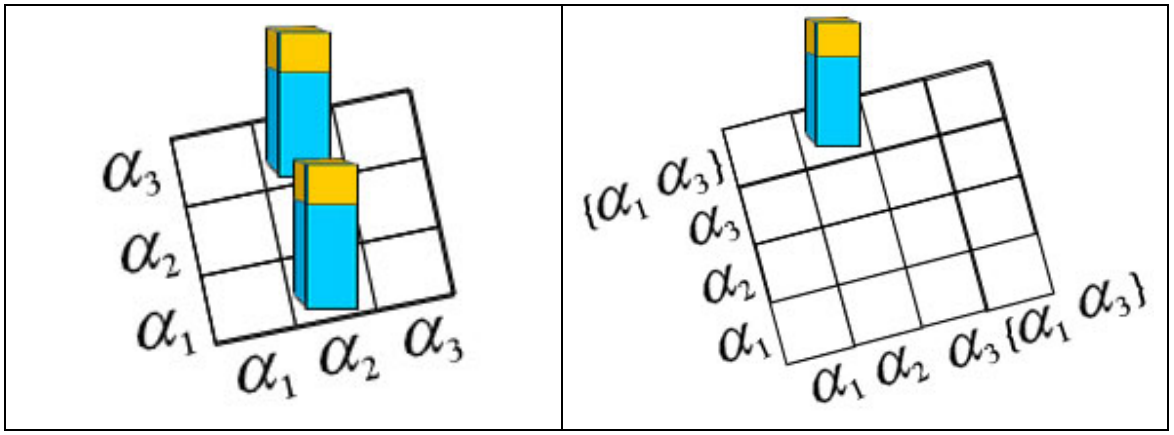


Figura 4.1. Matriz bidimensional.
Antecedentes en eje vertical, consecuentes en eje horizontal

Si, por otro lado, consideramos reglas cuyo antecedente esta representado por un conjunto de gramas, existen dos representaciones distintas para la matriz bidimensional [41]:

$$R_2 : \{\alpha_1, \alpha_3\} \rightarrow \alpha_2$$



A **B**

Figura 4.2. Matriz bidimensional con múltiples antecedentes
Antecedentes en eje vertical, consecuentes en eje horizontal

Se puede observar en la figura 4.2 que el esquema representado en la matriz B es mucho más representativo que el que se muestra en la matriz A, debido a que, en esta última, no se distingue si se trata de una regla o dos.

4.6. b. Grafos Dirigidos

Los grafos dirigidos son otra de las formas para representar una regla de asociación. Este método utiliza la condición natural de los grafos para representar relaciones entre objetos. Cada uno de los vértices del mismo corresponde a un ítem del conjunto de elementos que pueden formar parte de los antecedentes o consecuentes de la regla y el arco dirigido representa la relación entre ellos (regla de asociación). Según [41] este método es útil sólo para reglas que impliquen pocos ítems debido a que se dificulta considerable la lectura del mismo.

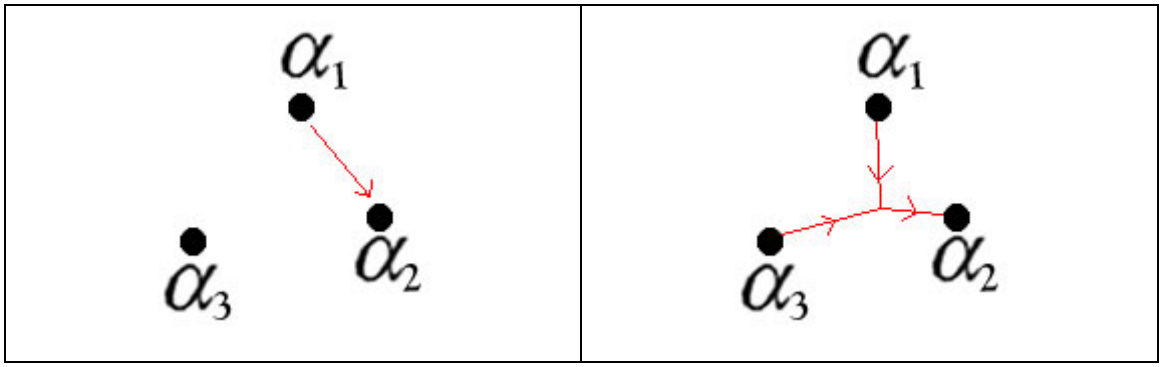
Siguiendo la línea de la sección anterior, Para ejemplificar el funcionamiento de este método utilicemos las mismas reglas mencionadas anteriormente:

$$R_1 : \alpha_1 \rightarrow \alpha_2$$
$$R_2 : \{\alpha_1, \alpha_3\} \rightarrow \alpha_2$$

Con el alfabeto,

$$\sum = \{\alpha_1, \alpha_2, \alpha_3\}$$

Los grafos resultantes son los siguientes:



A **B**

Figura 4.3. Grafo dirigido

A) Un solo antecedente regla R_1

B) Varios antecedentes regla R_2

Como puede verse el grafo dirigido puede convertirse en un gran conjunto de arcos y nodos difícil de leer al incrementarse el número de ítems y reglas.

CAPITULO 5

Definición del Modelo

Como se mencionó en capítulos anteriores, esta tesis tiene como fin presentar un algoritmo basado en la aplicación del modelo de agentes a la teoría de grafos para la detección de patrones en textos. En este capítulo se pretende interiorizar en la enunciación de la estructura de datos y los algoritmos de selección y búsqueda necesarios para cumplir con dicho objetivo.

En la primera sección, se define formalmente el grafo que modeliza a un texto para la detección de patrones. Básicamente, cada nodo del grafo representa un grama del texto, mientras que los arcos simbolizan la relación existente entre ellos. El grafo que se especifica se puede categorizar como un atributed graph, ponderado y dirigido.

Por otro lado, con el fin de reducir el espacio de búsqueda y agilizar el proceso de detección de patrones , se presenta, en la segunda sección de este capítulo, un método para la selección de los arcos más relevantes del nodo, el cual permite definir un subgrafo de menor tamaño que contenga sólo aquellos arcos más significativos del grafo original. Para poder llevar adelante dicha tarea mientras se procesa el texto, se consideró a cada uno de los nodos del grafo como un agente cuya meta es la de maximizar el promedio del peso de los arcos que parten de él, dicha modelización se encuentra plasmada en la tercera sección de este capítulo.

Finalmente, en la última sección se presenta un resumen que pretende integrar todos los conceptos fijados en este apartado. Todo lo especificado en este capitulo hace referencia a lo determinado en [11].

5.1. Estructura de datos

5.1.1 Definición del grafo

Los grafos son estructuras útiles para esquematizar relaciones entre elementos. Es debido a esto que, para la modelización del texto del cual se quiere extraer patrones, se utiliza un grafo donde los gramas sean nodos y la relación entre ellos, arcos.

Como se puntualizó en capítulos anteriores, un grafo es una tripla ordenada $(V(G), E(G), \psi_G)$, y la correcta formalización del grafo depende de la definición de la misma.

En principio, se considera a un nodo como el representante de un grama del texto. Dicho grama posee una frecuencia de aparición, la cual es considerada como un atributo del nodo. Según lo dicho, $V(G)$ se define de la siguiente forma:

Def. 5.1: *Sea* Σ *un alfabeto cualquiera, entonces* V *es un espacio definido por el producto cartesiano* $V : \Sigma x \Re$ *, donde* $\Re$ *representa el espacio de los números reales.*

Por lo tanto, cuando se analiza un texto T_i para la detección de patrones, un nodo $v \in V$ se puede representar como el vector $v = [\alpha, f]$, siendo α un elemento del alfabeto Σ (un grama) y $f \in \Re$ la frecuencia de aparición de dicho grama en T_i, dada por la ecuación:

$$f = \frac{N_\alpha}{N_t} \qquad [5.1]$$

Donde N_α es el número de instancias de α en T_i, y N_t el número total de instancias en T_i.

Por otro lado, para detectar un patrón de dos gramas debe existir una dependencia entre ellos, la cual debe ser posible de observar en el texto con una frecuencia relativamente alta. En nuestro caso, un arco representa una relación ponderada entre dos nodos, por lo que, representa claramente la dependencia antes mencionada. El conjunto de arcos se puede definir de la siguiente manera:

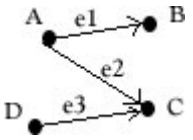
Def. 5.2: *E es un espacio definido por el producto cartesiano* $E : VxVx\Re$.

A partir de esta definición se interpreta que, sea $e \in E, e$ se puede representar como $e = [v_1, v_2, w]$, siendo $v_1, v_2 \in V$ y $w \in \Re$, el peso del arco).

Finalmente, siguiendo la nomenclatura expresada en [5], el último elemento de la tripla que resta definir es la función ψ_G que vincula dos nodos a un arco. Sean $e \in E, u, v \in V$ y $\psi_G(e) = uv$ se

dice que e une a u y v, y éstos son extremos de e. En nuestro caso, esta función puede leerse como "v *antecede a* u *en el texto*". Como se dijo en el capitulo 2, la función ψ_G ofrece una nomenclatura para definir las aristas del grafo y, en caso de los grafos dirigido, la dirección de sus arcos.

Ejemplo:



$$V = \{A, B, C, D\}; E = \{e_1, e_2, e_3\}$$
$$\psi_G(e_1) = AB : \text{"B antecede a A en el Texto"}$$
$$\psi_G(e_2) = AC : \text{"C antecede a A en el Texto"}$$
$$\psi_G(e_3) = DC : \text{"C antecede a D en el Texto"}$$

Es necesario mencionar las siguientes propiedades de ψ_G:

1) Inyectividad:

$e_1, e_2 \in E$, $u, v \in V$, $\psi_G(e_1) = \psi_G(e_2) \leftrightarrow e_1 = e_2$

2) Grafo Dirigido:

Sean $e_1, e_2 \in E$, $u, v \in V$, $u \neq v$, $e_1 \neq e_2$, $\psi_G(e_1) = uv$ y $\psi_G(e_2) = vu \rightarrow \psi_G(e_1) \neq \psi_G(e_2)$

Ya definida la tripla, sólo resta formalizar la definición del grafo que se utilizará a lo largo del documento:

Def. 5.3 (grafo del modelo): *se define* $H = [V_H, E_H, \psi_G]$ *como un grafo dirigido, donde* $V_H \subset V$, $E_H \subset E$.

5.1.2. Función peso ($\xi(d)$)

Los patrones en la red se detectan a partir de los valores de peso de los arcos del grafo. Este valor es modificado según la distancia que se observa entre dos gramas de un texto. La función peso es la que pondera dicha distancia. No es menester de esta tesis fijar una expresión analítica de la función, debido a que ésta se encuentra sujeta al problema que se desea solucionar.

La función peso también juega un papel preponderante en el número de arcos y nodos de la red, debido a que la ponderación de la distancia es utilizada por el mecanismo de selección para la eliminación de arcos *[sección 5.2]*.

Debido a que el propósito de esta sección es presentar el objetivo de la función peso, no se define una expresión analítica de ésta. Sin embargo, se expresan las siguientes restricciones

$$\xi(d) \geq 0 \forall d \in \Re^+ \qquad [5.2]$$
$$\exists \xi(d) \forall d \in N^+$$

Es ministerio de quién implemente el modelo que se presenta en este capítulo el definir la expresión analítica de $\xi(d)$ acorde a los objetivos que desee cumplir.

Para aclarar el uso de la función peso, abordemos el siguiente texto:

Ejemplo 5.1:

Consideremos los siguientes parámetros del algoritmo:

Texto a procesar: "La iteración es humana; la recursión divina"

$$\xi(d) = \frac{1}{d+1}$$

$$\Sigma = \{La, iteración, es, humana, recursión, divina\}$$

NOTA: Si bien el objetivo de este ejemplo no es definir una función $\xi(d)$ para una finalidad especial, se puede observar que la misma es una función que pondera las distancias pequeñas con mayor peso. De esta forma, la relación entre las palabras que se encuentran cercanas entre sí tendrá una ponderación mayor que las que se encuentran alejadas. Así se ponderan de mejor forma a las palabras que se encuentren, por ejemplo, en la misma oración.

Cuando se analice el grama humana se tomaran en consideración los siguiente pesos:

Relación "La antecede a humana en el texto": $\frac{1}{3+1} = \frac{1}{4}$

Relación "iteración antecede a humana en el texto": $\frac{1}{2+1} = \frac{1}{3}$

Relación "es antecede a humana en el texto": $\frac{1}{1+1} = \frac{1}{2}$

Observar que las distancias presentadas (3, 2, 1) son las distancias observadas entre el grama "humana" y "La", "iteración", "es", respectivamente.

5.1.3. Actualización del grafo

Conforme se recorre un texto para la detección de patrones, el grafo que modeliza dicho documento debe cambiar. El algoritmo que se presenta en esta tesis no parte de una estructura de datos definida sino que actúa cada vez que se detecta un grama, modificando y extendiendo al grafo.

La actualización del grafo se puede dar por dos causas:

a) La aparición de un nuevo grama en el texto.

b) La aparición un grama preexistente.

5.1.3.a Aparición de un nuevo grama en el texto (ANGT)

Debido a que un grama sucede a otro, existe un relación entre el nuevo nodo y los ya presentes en el grafo, es por ello que se debe crear un arco entre el nuevo grama y cada unos de los gramas predecesores.

Formalmente, sea $H^n = (V_H^n, E_H^n)$ un estado n del grafo y $v \in V$ un nuevo vértice, la actualización del grafo se lleva adelante mediante la siguiente función:

$$\Lambda(v) = \bigcup_{v_i \in V_H^n} (v, v_i, t(x_i)) \qquad [5.3]$$

Siendo t la función definida como $t : \Re^k \rightarrow \Re$

$$t(x_i) = \sum_{j=1}^{k} \xi(x_{i\,j}) \qquad [5.4]$$

xi: vector de distancias entre el grama v y vi

k: numero de instancias anteriores a v

Se puede observar que $\Lambda(v)$ genera el conjunto de todos los arcos nuevos, mientras que $t(xi)$ realiza la sumatoria de todos los pesos existentes entre la aparición del grama nuevo y las instancias de vi anteriores a dicha aparición.

Debido a que este modelo está diseñado para una aplicación práctica, se debe hacer mención al valor de k. Dicho número, representa la cantidad de instancias v_i anteriores a v. No obstante, al implementar en un software el algoritmo de actualización, se encuentra inmanejable, en cuestiones de performance y memoria, el almacenamiento de dichas instancias. Es debido a esto que, se introduce al modelo el concepto de *ventana de visualización de gramas* (σ). Dicha *ventana* fija el número de gramas previos a una actualización del grafo. Este concepto puede ser entendido más fácilmente con el siguiente ejemplo:

Ejemplo 5.2.

Supongamos el siguiente texto:

"Ansí, han de saber que Dios no crío cantidá ninguna. El ser de todos los seres sólo formó la unidá; Lo demás lo ha criado el hombre después que aprendió a contar."

Y supongamos un tamaño de ventana σ =3. En el momento en que se analice el grama "de", que se encuentra subrayado, sólo se tomarán en cuenta, para la actualización del grafo, las distancias existentes entre:

- *"de" y "ser"*
- *"de" y "El"*
- *"de" y "ninguna"*

Las cuales entran en la ventana de visualización de gramas. Luego, al procesarse el grama "todos", la palabra "ninguna" no será tomada en cuenta para la actualización del grafo, por no pertenecer a la ventana de visualización de gramas.

El valor de σ no es determinado por el modelo, sino que es fijado por quién implemente la red acorde al problema y a la función peso que se este utilizando.

Resta mencionar, aunque trivialmente, que *k* es siempre menor o igual a σ. Dicha restricción hace posible el manejo computacional de la función $t(x_i)$.

Finalmente,

$$H^{n+1} = (V_H^{n+1}, E_H^{n+1}) \qquad [5.5]$$

Donde,

$$V_H^{n+1} = V_H^n \bigcup v \qquad \text{y} \qquad E_H^{n+1} = E_H^n \bigcup \Lambda(v) \qquad [5.6]$$

5.1.3.b. Aparición grama preexistente (ANE)

Cada vez que aparezca un nodo ya presente en el grafo, se llevará adelante la operación de actualización de pesos que formalmente se define como:

$$\text{Sea } v \in V_H^n, \forall e \in \bigcup_{vi}^{+}, w(e) = w(e) + t(x_{vi}) \qquad [5.7]$$

Donde,

x_{vi} : vector de distancias entre dos nodos.

w : peso del arco e

$\bigcup_{vi}^{+}$: conjunto de arcos salientes de v

A igual que en la sección anterior, es necesario remarcar que la definición presentada sólo es válida en un marco teórico, debido a que dicha actualización es viable computacionalmente sólo sí se tiene en cuenta la *ventana de visualización de gramas* que limita el tiempo y memoria necesaria para implementar el algoritmo.

5.2. Método de selección

5.2.1. Objetivo

El método de selección que se presenta en este apartado tiene como finalidad reducir el espacio de búsqueda de los patrones de gramas. Este algoritmo permite eliminar aquellos arcos que posean un peso bajo con respecto a la sumatoria total de los pesos del conjunto de arcos salientes del nodo. De esta forma, se eliminan caminos innecesarios y se obtiene un grafo de menor tamaño que facilita el proceso de búsqueda de patrones. Formalmente, el objetivo del algoritmo se puede definir de la siguiente manera:

Def 5.4. *Sea* $H = (V, E)$ *un grafo ponderado con función de peso* $w : E \to \Re$, *encontrar un subgrafo* $H' = (V', E') / \ \forall v \in V', w(v) > w_{\min}$, *donde* $w_{\min}$ *es una constante del problema.*

5.2.2. Algoritmo y Estructura de Datos

Se define, en principio, el conjunto de arcos $A = \{a_1, a_2, ..., a_n\}$ donde $A = \bigcup_v^+$, $v \in V$. Cada arco de A será evaluado con respecto al conjunto según la función probabilística:

$$\overline{w}^m{}_i = \frac{w^m(a_i)}{\sum_{a_j \in A} w^m(a_j)} \qquad [5.8]$$

Donde, $w^m(a_j)$ es el peso del arco a_j en una instancia *m* del nodo *v*.

Luego, se especifica $[\overline{w}_{\min}; \overline{w}_{\max}]$ como el rango de valores posibles para $\overline{w}_i^m$. Es decir, se considera que todo arco cuyo $\overline{w}_i^m$ sea menor a $\overline{w}_{\min}$ no debe ser tomado en cuenta y por ende puede ser eliminado. Mientras que todo arco que tenga un $\overline{w}_i^m$ mayor a $\overline{w}_{\max}$ puede ser interpretado como una relación fuerte entre gramas y en consecuencia un patrón.

Para mantener los arcos ordenados por peso y así agilizar el proceso de selección de arcos, cada arco de A es dispuesto en un vector Ω de q posiciones según su $\overline{w}_j^m$ de la siguiente forma:

$$a_j \in \Omega(i) \leftrightarrow \left(\overline{w}_{\min} + (i-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} \right) \right) < \overline{w}^m{}_j < \left(\overline{w}_{\min} + i.\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} \right) \right); 1 \le i < q$$

ó

$$a_j \in \Omega(q) \leftrightarrow \left(\overline{w}_{\min} + (q-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} \right) \right) < \overline{w}^m{}_j \qquad [5.9]$$

> ***Nota****: Las restricciones determinadas anteriormente son sólo de carácter teórico para la definición del algoritmo, para una mejor determinación de las mismas referirse a sección 6.1.*

Cada vez que se genera una instancia del nodo v (aparece el grama en el texto) se actualizan todos los pesos de los arcos [*ver sección 5.1.3.a y 5.1.3.b*], y en consecuencia los $\overline{w}^m{}_j$. Esto puede generar que, los arcos activados incrementen su posición en el vector. Debido a la presencia de la ventana de visualización de gramas [*ver sección 5.1.3.a*] cada vez que sucede una instancia del nodo v puede haber algunos arcos que no se activen y por ende su posición en el vector puede disminuir. Es decir, cada vez que aparezca un

grama en el texto se debe evaluar a cada uno de los arcos del nodo para verificar su correcta posición en el vector. Debido a que esto es costoso en tiempo, se decidió implementar la siguiente solución: Sucedido un tiempo ε [*ver sección 5.2.3*] el nivel inferior se limpiará de arcos (se eliminan) y éste pasará a ser el nivel superior del vector. De esta forma, terminada la lectura del texto, sólo quedarán los arcos con mayor peso en el nivel q del vector.

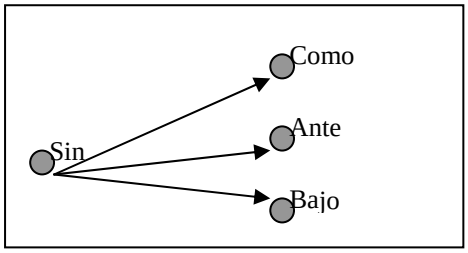
Por otro lado, cada posición del vector tiene lo que se denomina una zona segura, todo elemento perteneciente a esa zona no descenderá de nivel cuando pase el tiempo ε. La pertenencia de un elemento del vector a la zona de seguridad se explicará en la sección 5.2.4.

Finalmente, se debe mencionar que el valor de q, como así también, el rango $\left[\overline{w}_{\min}; \overline{w}_{\max}\right]$ son parámetros del algoritmo, en consecuencia deben ser definidos por quién lo implemente.

A continuación, se presenta un ejemplo para aclarar el funcionamiento del método de selección:

Ejemplo 5.3

Para poder ejemplificar el algoritmo de selección debemos determinar las constantes del problema. Para que el ejemplo sea simple se considerará el siguiente grafo:



Por otro lado, no se evaluará un texto real, sino que se presentará un conjunto de modificaciones a los pesos de los arcos antes graficados.

Supongamos, $q = 3$; $\overline{w}_{\min} = 0{,}1$; $\overline{w}_{\max} = 0{,}6$; $\sigma = 3$; $\varepsilon = 4$.

Cada instancia del nodo "Sin" se representará en el ejemplo se la siguiente manera:
(Número de instancia, {(grama antecesor 1, peso 1), (grama antecesor 2, peso 2), …. (grama antecesor n, peso n)})

Definimos entonces la siguiente muestra de instancias de nodo "Sin":
(1,{ ("ante",3)})
(2,{("bajo",1)})
(3,{("bajo",4)})
(4,{("bajo",6)})
(5,{("ante",7),("bajo",6),("ante",5)})
(6{("ante",4)})
(7,{("bajo",4)})
Y suponemos el siguiente estado inicial de Ω*:*

3	Bajo (10) (0,5)	$0{,}43 \le \overline{w} < 0{,}6$
2	Ante (7) (0,35)	$0{,}26 \le \overline{w} < 0{,}43$
1	Como (3) (0,15)	$0{,}1 \le \overline{w} < 0{,}26$
Total ponderación: 20		

El cuadro se lee de la siguiente manera:

El arco que une a "Sin" y a "Bajo" tiene peso 10 y un valor de $\overline{w}$ *igual a 0,5. Se encuentra en el nivel 3 del vector donde se hallan los arcos cuyo* $\overline{w}$ *es mayor a 0,43 y menor a 0,6.*
La lectura de los otros niveles del vector es análoga a la dada.

Fijadas todas las constantes del problema, comenzamos a emular el algoritmo:

- ***(1,{ (“ante”,3)})***

3	Bajo (10) (0,43) Ante (10) (0,43)	$0,43 \leq \overline{w} < 0,6$
2		$0,26 \leq \overline{w} < 0,43$
1	Como (3) (0,13)	$0,1 \leq \overline{w} < 0,26$
Total ponderación: 23		

El arco“Ante” fue activado, aumentó su peso como así también su valor de $\overline{w}$. Se evaluó su pertenencia al nivel 3 y aumento su posición.

- ***(2,{(“bajo”,1)})***

3	Bajo (11) (0,45) Ante (10) (**0,41**)	$0,43 \leq \overline{w} < 0,6$
2		$0,26 \leq \overline{w} < 0,43$
1	Como (3) (0,12)	$0,1 \leq \overline{w} < 0,26$
Total ponderación: 24		

El arco “Ante” no debería estar en el nivel 3, sin embargo, no fue activado en esta fase por lo que no se verifica su posición. Por lo tanto, permanece en el nivel a pesar de su peso.

- ***(3,{(“bajo”,4)})***

3	Bajo (15) (0,53) Ante (10) (**0,35**)	$0,43 \leq \overline{w} < 0,6$
2		$0,26 \leq \overline{w} < 0,43$
1	Como (3) (0,10)	$0,1 \leq \overline{w} < 0,26$
Total ponderación: 28		

El arco “Ante” permanece en la misma situación que en el caso anterior.

- ***(4,{(“bajo”,6)})***

3	Bajo (21) (0,61) Ante (10) (**0,29**)	$0,43 \leq \overline{w} < 0,6$
2		$0,26 \leq \overline{w} < 0,43$
1	Como (3) (**0,08**)	$0,1 \leq \overline{w} < 0,26$
Total ponderación: 34		

Se observa ahora que el valor de $\overline{w}$ del arco “Como” se encuentra por debajo de lo permitido. Sin embargo, no fue activado por lo que su peso no se evalúa.

Limpio (1)

1	Bajo (21) (0,61)	$0,43 \leq \overline{w} < 0,6$
3	Ante (10) (0,29)	$0,26 \leq \overline{w} < 0,43$
2		$0,1 \leq \overline{w} < 0,26$
Total ponderación: 34		

Sucedieron 4 instancias del nodo "Sin", por lo tanto se limpia la última posición del vector. Recordar que 4 es el valor de ε.
El arco que representa la conexión entre "Sin" y "Bajo", no disminuye de nivel por su peso.

- ***(5,{("ante",7),("bajo",6),("ante",5)})***

1	Bajo (27) (0,51)	$0,43 < \overline{w} < 0,6$
3	Ante (22) (0,42)	$0,26 < \overline{w} < 0,43$
2		$0,1 < \overline{w} < 0,26$
Total ponderación: 52		

Se modifican los pesos de los arcos. Sin embargo no hay redistribución.

- ***6{("ante",4)})***

1	Ante (26) (0,49) Bajo (27) (0,50)	$0,43 < \overline{w} < 0,6$
3		$0,26 < \overline{w} < 0,43$
2		$0,1 < \overline{w} < 0,26$
Total ponderación: 53		

Se activa el arco "Ante", esto genera que se actualice su peso y sea evaluada su posición en el vector.

- ***(7,{("bajo",4)})***

1	Ante (26) (0,45) Bajo (31) (0,54)	$0,43 < \overline{w} < 0,6$
3		$0,26 < \overline{w} < 0,43$
2		$0,1 < \overline{w} < 0,26$
Total ponderación: 57		

Se activa el arco "Bajo", esto genera que se actualice su peso y sea evaluada su posición en el vector.

El ejemplo anterior no pretende ser una representación de una situación real sino que sus constantes y valores fueron elegidos para visualizar todas las variantes posibles. Se puede observar que uno de los factores críticos que se debe resolver para no tener inconsistencias en el vector es el valor de ε. Dicho valor no es definido por quién implemente el modelo sino que es una constante del mismo y será calculado en la siguiente sección.

5.2.3. Definición del tiempo ε

El objetivo de esta sección es establecer los valores de ε para poder completar la definición del método de selección de arcos. Básicamente, se determina una sucesión matemática que permite predecir el número de instancias de un nodo que se deben esperar para que aquellos arcos que no hayan sido activados disminuyan un nivel en el vector Ω *[sección 5.2.2]*.

Cada vez que se activa un nodo del grafo (instancia *m*) existe un conjunto de gramas en la ventana de visualización. Cuando se actualizan los pesos de los arcos salientes del nodo, sólo se tienen en cuenta dichos gramas. Las distancias existentes entre el nodo y cada uno de ellos determinan un conjunto de incrementos de pesos que denominamos P_{vv}^{m}.

$$P_{vv}^{m} = \left\{p_1^m, p_2^m, ..., p_\sigma^m\right\}$$

$$P_{vv}^{m+} = \sum_{i=1}^{\sigma} p_i^m \qquad \textit{[5.10]}$$

Por otro lado, cada nodo *v* del grafo tiene asociado un conjunto de arcos $A = \bigcup_v^+ = \{a_1, a_2, ..., a_n\}$. Cada elemento de este conjunto será comparado mediante la expresión *5.8.* Si tomamos en cuenta dicha expresión y si no hay aumento del peso del arco a_i $\left(w^{m+1}(a_i) = w^m(a_i)\right)$se observa que:

$$\overline{w}_i^{m+1} = \frac{w^m(a_i)}{\sum_{a_j \in A} w^m(a_j) + P_{vv}^{(m+1)+}} \qquad \textit{[5.11]}$$

Es decir, el peso en el estado $m+1$ es igual a el peso en el estado m (no se activa e arco a_i) sobre el total de pesos en el estado m más el incremento de pesos generados a partir de la suma de pesos en la ventana de visualización (expresión *5.10*).

Si suponemos $P_{vv}^{m+} = P_{vv}$ constante, luego de ε instancias sin incremento de peso en el arco a_i, se tiene:

$$\overline{w}_i^{m+\varepsilon} = \frac{w^m(a_i)}{\sum_{a_j \in A} w^m(a_j) + \varepsilon.P_{vv}} \qquad \textit{[5.12]}$$

Finalmente, otro elemento a tener en cuenta es la posición del arco en el vector Ω. Como se mencionó en la sección *5.2.2*, el arco a_i se encuentra en el vector Ω según la forma *5.9.* Siguiendo dicha expresión

se puede observar que el valor de peso máximo de cada nivel b del vector Ω puede expresarse de la siguiente manera:

$$\theta(b) = \overline{w}_{\min} + \frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} b \qquad [5.13]$$

Entonces, la variación de peso entre un nivel y otro es….

$$\Lambda(b) = \theta(b) - \theta(b-1)$$

$$\Lambda(b) = \overline{w}_{\min} + \frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} b - \left(\overline{w}_{\min} + \frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} (b-1) \right) =$$

$$= \left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} \right)$$

Se observa que $\Lambda(b)$ es independiente de *b*, por lo tanto…

$$\Lambda = \Lambda(b) = \left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q} \right) \qquad [5.14]$$

Ya explicitados todos los elementos necesarios, se estudia cuántas instancias del nodo deben ocurrir para que un arco que se encuentre en un nivel determinado del vector disminuya su posición.

Sea τ el valor que debe disminuir el arco a_i para bajar un nivel en el vector y si se supone P_{vv}^{m+} constante se tiene:

$$\tau = \overline{w}_i^m - \overline{w}_i^{m+\varepsilon} \qquad [5.15]$$

Luego,

$$\tau = \frac{w^m(a_i)}{\sum_{a_j \in A} w^m(a_j)} - \frac{w^m(a_i)}{\sum_{a_j \in A} w^m(a_j) + \varepsilon . P_{vv}} = \frac{w^m(a_i) . \varepsilon . P_{vv}}{\varepsilon . P_{vv} (\sum_{a_j \in A} w^m(a_j)) + (\sum_{a_j \in A} w^m(a_j))^2}$$

$$\varepsilon = \frac{(\sum_{a_j \in A} w^m(a_j))^2 . \tau}{P_{vv}(w^m(a_i) - \tau \sum_{a_j \in A} w^m(a_j))} = \frac{\tau . \sum_{a_j \in A} w^m(a_j)}{P_{vv}(\overline{w}_i^m - \tau)}$$

Finalmente,

$$\varepsilon = \frac{\tau . \sum_{a_j \in A} w^m(a_j)}{P_{vv}(\overline{w}_i^m - \tau)} \qquad [5.16]$$

Si el arco a_i se encuentra en la posición b del vector, entonces τ puede escribirse de la siguiente manera:

$$\tau = \overline{w}_i^m - \theta(b-1) \qquad [5.17]$$

Luego,

$$\varepsilon = \frac{(\sum_{a_j \in A} w^m(a_j))(\overline{w}_i^m - \theta(b-1))}{P_{vv}(\overline{w}_i^m - (\overline{w}_i^m - \theta(b-1)))} \qquad [5.18]$$

$$\varepsilon = \frac{(\sum_{a_j \in A} w^m(a_j))(\overline{w}_i^m - \theta(b-1))}{P_{vv}.\theta(b-1)}$$

Debido a que la eliminación de los arcos y el descenso de un arco de nivel en el vector Ω se lleva a cabo siempre que sucedan ε iteraciones del nodo y no a partir de la evaluación de cada uno de los arcos, se decide esperar lo máximo posible antes colocar a un arco en un nivel inferior, de esta forma se evita que un arco permanezca en un nivel menor al definido por su peso. El objetivo principal de esta norma es que la disposición de los arcos cumpla con el límite superior determinado en la expresión *5.9*. De esta forma, se elige

$$\theta(b-1) = \overline{w}_{\min}$$

$$\overline{w}_i^m - \theta(b-1) = \Delta$$

Por último se define β_m como la relación entre la sumatoria de los pesos de los arcos eliminados y el total de pesos asociados al nodo.

$$\beta_m = \frac{\sum_{\Gamma m} w^m(a_i)}{\sum_{a_j \in A} w^m(a_j)} \qquad [5.19]$$

Donde Γm es el conjunto de pesos de los arcos eliminados en la instancia m. Así, para la primera iteración:

$\sum_{a_j \in A} w^1(a_j) = P_{vv}$, y $\beta_0 = 0$ (no se elimina ningún arco)

Luego,

$$\varepsilon_1 = \frac{\Lambda}{\overline{w}_{\min}}$$

Para la segunda iteración se tiene:

$$\sum_{a_j \in A} w^2(a_j) = \varepsilon_1 P_{vv} - \beta_1 \varepsilon_1 P_{vv}$$

(Todo lo acumulado en la primera iteración menos lo eliminado)

Por lo tanto,

$$\varepsilon_2 = \varepsilon_1^2 (1 - \beta_1)$$

Siguiendo el mismo razonamiento se tiene:

$$\varepsilon_3 = \frac{\Lambda}{\overline{w}_{\min} P_{vv}} \left(\sum_{a_j \in A} w^{\varepsilon_1 + \varepsilon_2}(a_j) + \varepsilon_2 P_{vv} \right) (1 - \beta_2) = (\varepsilon_2 + \varepsilon_2 \varepsilon_1)(1 - \beta_2)$$
$$\varepsilon_3 = \varepsilon_2 (1 + \varepsilon_1 (1 - \beta_2) - \beta_2)$$

$$\varepsilon_4 = (\varepsilon_3 + \varepsilon_3 \varepsilon_1)(1 - \beta_3)$$
$$\varepsilon_4 = \varepsilon_3 (1 + \varepsilon_1)(1 - \beta_3) = \varepsilon_3 (1 + \varepsilon_1 - \beta_3 - \beta_3 \varepsilon_1)$$
$$\varepsilon_4 = \varepsilon_3 (1 + \varepsilon_1 (1 - \beta_3) - \beta_3)$$

$$\varepsilon_5 = (\varepsilon_4 + \varepsilon_4 \varepsilon_1)(1 - \beta_4)$$
$$\varepsilon_5 = \varepsilon_4 (1 + \varepsilon_1)(1 - \beta_4) = \varepsilon_4 (1 + \varepsilon_1 - \beta_4 - \beta_4 \varepsilon_1)$$
$$\varepsilon_5 = \varepsilon_4 (1 + (1 - \beta_4)\varepsilon_1 - \beta_4)$$

Para todo $n > 2$, se tiene…

$$\varepsilon_n = \frac{\Lambda}{\overline{w}_{\min} P_{vv}} \left(\sum_{a_j \in A} w^{\sum_{i=1}^{n-1} \varepsilon_i}(a_j) + \varepsilon_{n-1} P_{vv} \right) (1 - \beta_{n-1})$$
$$\varepsilon_n = (\varepsilon_{n-1} + \varepsilon_{n-1} \varepsilon_1)(1 - \beta_{n-1})$$
$$\varepsilon_n = \varepsilon_{n-1} (1 + \varepsilon_1)(1 - \beta_{n-1}) = \varepsilon_{n-1} (1 + \varepsilon_1 - \beta_{n-1} - \beta_{n-1} \varepsilon_1)$$
$$\varepsilon_n = \varepsilon_{n-1} (1 + (1 - \beta_{n-1})\varepsilon_1 - \beta_{n-1})$$

En resumen....

$$\varepsilon_1 = \frac{\Lambda}{\overline{w}_{\min}}$$
$$\varepsilon_2 = \varepsilon_1^2(1-\beta_1) \qquad [5.20]$$
$$\varepsilon_n = \varepsilon_{n-1}(1+(1-\beta_{n-1})\varepsilon_1)$$

5.2.4. Zona de seguridad

Cada posición del vector Ω posee lo que se denomina una zona de seguridad. Todo arco que pertenezca a dicha zona se llama *"arco seguro"*, sin embargo aquel arco que no se encuentre en una zona de seguridad se designa como *"no seguro"*. El manejo de las zonas de seguridad se rige mediante los siguientes reglas:

1. Todo arco que se encuentre en una zona de seguridad no desciende de nivel cuando se eliminan los arcos de la última posición de Ω.
2. Cada vez que se produce una eliminación del nivel inferior todo arco perteneciente una zona de seguridad pierde la condición de *"arco seguro"*, es decir, sale de la zona segura.
3. Siempre que se produzca una eliminación del nivel inferior, toda zona segura se completará con los arcos *"no seguros"* del nivel inmediatamente superior.

El funcionamiento de las zonas de seguridad puede visualizarse fácilmente en la figura *5.2*.

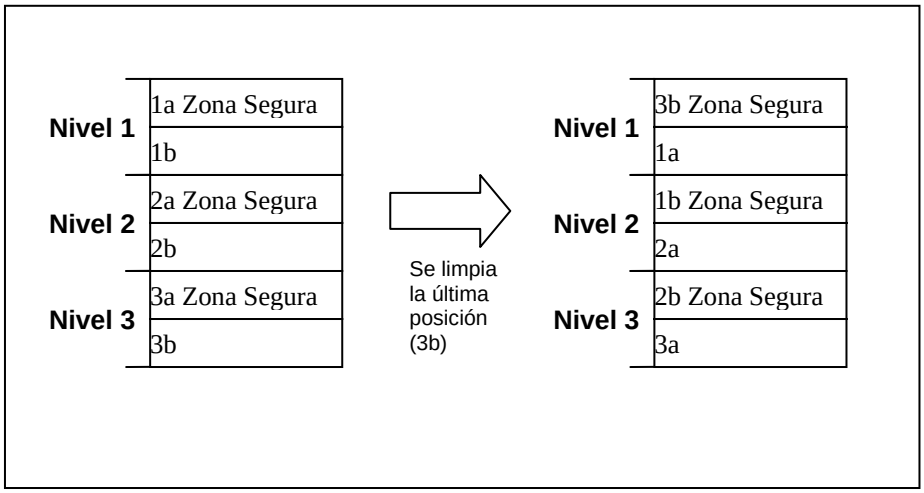


Figura 5.2. Zonas seguras

En esta sección se define cuánto debe ser el valor del incremento del peso de un arco para entrar en una zona de seguridad. Para ello se sigue el siguiente razonamiento: supongamos que en el instante *m* se definen las ε iteraciones que se deben esperar para realizar una nueva eliminación de arcos. Dicho valor

se calcula para aquellos arcos que no sean activados en todo el período, es decir, no incrementen su valor peso. Sin embargo, no todos los arcos cumplen con esta condición, es por ello que suponemos un instante t menor a ε donde se produce la activación de un nodo a_i perteneciente al nivel *b* del vector Ω. La siguiente figura esquematiza la situación:

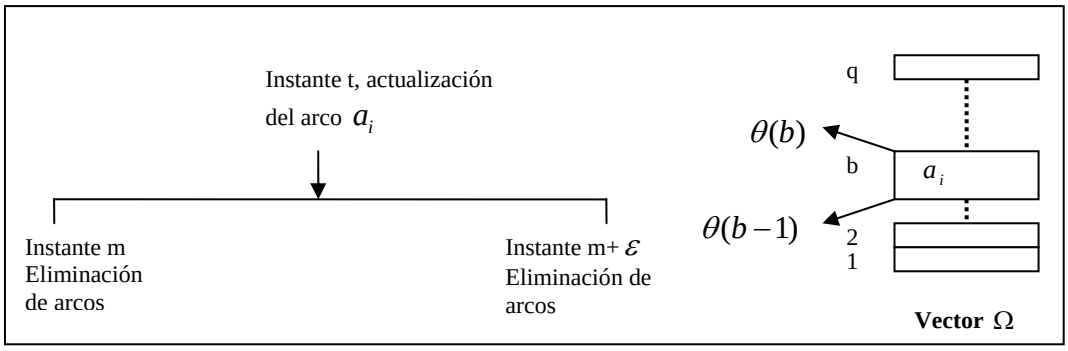


Figura 5.3. Situación gráfica de la actualización de un arco.

Un elemento que se encuentra en una la de seguridad de un nivel b no desciende de nivel al momento de la eliminación de los arcos. Es así que, para que un elemento del vector entre en una zona de seguridad su relación de peso con el total en el instante $t(w_i^t)$ debe ser tal que en el instante m+ ε (instante de eliminación) su peso $\overline{w}_i^{m+\varepsilon}$ debe ser mayor al límite inferior del nivel donde se encuentra, es decir:

$$\theta(b-1) \le \frac{w^{m+t-1}(a_i) + I^t}{W^{m+t-1} + P_{vv} + (\varepsilon - t)P_{vv}} \qquad [5.21]$$

Siendo

$$W^{m+t-1} = \sum_{a_i \in A} w^{m+t-1}(a_i)$$

I^t, el incremento del arco en el instante t

Definimos entonces el valor del incremento para entrar en la zona de seguridad:

$$\theta(b-1) \le \frac{w^{m+t-1}(a_i) + I^t}{W^m + tP_{vv} + (\varepsilon - t)P_{vv}}$$

$$\theta(b-1) \le \frac{w^{m+t-1}(a_i) + I^t}{W^m + \varepsilon P_{vv}}$$

$$\theta(b-1)(W^m + \varepsilon P_{vv}) - w^{m+t-1}(a_i) \le I^t \qquad [5.22]$$

Es decir, todo incremento en el peso de un arco que cumpla con la condición *5.22* provocará un pasaje del mismo a la zona de seguridad del nivel. Finalmente encontramos una expresión para la zona de seguridad de un nivel (θ_{seguro}):

$$\frac{w^{m+t-1}(a_i) + I^t}{W^{m+t}} \geq \theta_{seguro}$$

$$\frac{\theta(b-1)\left(W_j^m + \varepsilon P_{vv}\right)}{\left(W^m + t.P_{vv}\right)} \geq \theta_{seguro} \qquad [5.23]$$

Se puede observar que el tamaño de la zona de seguridad es inversamente proporcional al instante t, es decir, acorde suceden instancias del nodo, es más sencillo para un arco entrar a la zona de seguridad. La figura *5.4* muestra esta situación.

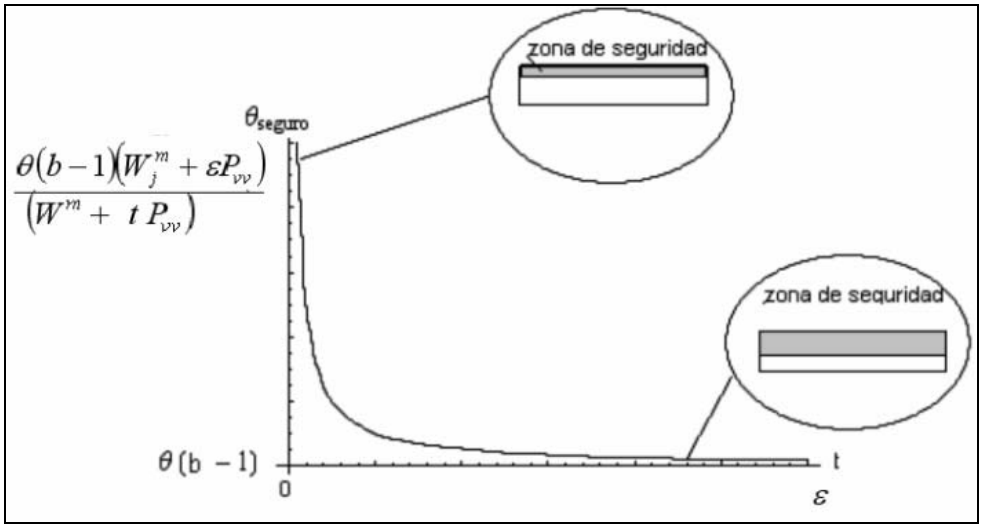


Figura 5.4. Tamaño de la zona de seguridad en función de t

Se puede ver en la grafica anterior que cada posición del vector posee una zona de seguridad distinta, sin embargo, la variación del tamaño de la misma posee la misma curva independientemente del nivel de Ω.

5.3. Modelo de Agente

Uno de los objetivos que se presenta en este documento es la reducción automática del volumen de arcos para una fácil manipulación del grafo y para ser lo más selectivo posible a la hora de detectar patrones. Para cumplir esta meta, se propone la aplicación del modelo de agente a los nodos del grafo planteando sus deseos, metas y planes. Muchos de los conceptos explicados en los apartados anteriores se resumen en esta sección pero enfocados desde el punto de vista del agente.

Desde el punto de vista global, el grafo puede entenderse como un conjunto de agentes simples interconectados entre sí. Con el término simples nos referimos a que cada nodo no lleva a cabo un grupo

complejo de acciones sino que actúa a partir de primitivas sencillas que permiten, a escala global, desarrollar una actividad compleja. La ínterconectividad de los agentes se asocia principalmente a los arcos del grafo, sin embargo, el accionar de cada vértice es independerte de resto por lo que, los nodos, no comparten información sobre el medio que los circunda. La comunicación interagentes se lleva a cabo sólo una vez terminado el procesamiento del texto. En éste caso, los agentes se comunican entre sí el conjunto de arcos de mayor peso que tienen en posesión.

El funcionamiento interno de los agentes se encuentra explicitado en los párrafos siguientes, donde se especifican las creencias, metas y planes de los mismos.

5.3.1. Creencias

Cada nodo del grafo tiene una visión distinta de su entorno. Es decir, un conjunto diferente de creencias. Es por ello que las variables que aquí se mencionan se encuentran asociadas a un nodo en particular, o lo que es lo mismo, no tienen carácter global en el grafo.

En principio, las variables asociadas al conocimiento del agente pueden diferenciarse en *variables de definición de frecuencias y variables de peso*. Las primeras están relacionadas al manejo del tiempo mientras que las segundas se distinguen por su utilidad en la comparación de arcos.

Existen, básicamente, dos variables de frecuencia. Por un lado, la variable λ_v que define el número de instancias del nodo *v* y ε_i que especifica el número de instancias que se debe aguardar para poder realizar una eliminación de arcos. El cálculo de dicha variable se desarrolló anteriormente en la sección *5.2.3*.

En lo concerniente a los arcos, cada vértice del grafo tiene conocimiento del conjunto de arcos salientes de él. Dicho conjunto lo denominamos *A*. Formalmente,

$$A = \bigcup_v^+ \qquad [5.24]$$

A partir de este conjunto se pueden definir el segundo tipo de variables, las variables de peso.

Se define como W_v^m a la sumatoria pesos de las instancias de *A*:

$$W_v^m = \sum_{a_i \in A} w^m(a_i) \qquad [5.25]$$

Se observa que cada W_v^m depende exclusivamente de los arcos y de la instancia del nodo.

Existe, por otro lado, otra variable de peso introducida en la sección *5.2.2*, particularmente mediante la ecuación *5.8*. La variable $\overline{w}_v^m$ depende de la instancia del nodo y es la utilizada para la comparación de los arcos.

Es necesario destacar que todo elemento perteneciente al conjunto *A* se distribuye en el vector Ω según lo especificado en la sección *5.2.2*.

Finalmente, cada nodo desconoce absolutamente el estado de su nodo vecino. Sólo finalizado el procesamiento los nodos se comunicarán unos con otros proveyendo a su vecino con los arcos de mayor peso.

5.3.2. Deseos

A diferencia de las creencias, todos los nodos del grafo poseen la misma meta. El objetivo principalmente radica en la maximización del promedio de los pesos de los arcos que salen de él.

Formalmente,

$$\max\left(\frac{W_v^m}{card\left(\bigcup_v^+\right)}\right)$$

Se puede observar que una buena heurística para el cumplimiento de este objetivo radica en la selección de las mejores instancias del conjunto *A*. De esta manera se busca eliminar los caminos irrelevantes del grafo y poder, por lo tanto, detectar los patrones del texto de forma más rápida.

5.3.3. Plan

El plan de cada agente se encuentra regido principalmente por sus variables de frecuencia. El procedimiento básico se resume en la eliminación de arcos irrelevantes según lo determinado en el apartado del proceso de selección *[5.2]*.

Cada instancia del nodo es determinada por la aparición en el texto del grama que representa. Siempre que esto ocurra se incrementara en uno a la variable λ_v y mientras que ésta sea distinta de ε_i solo se procederá a la actualización de los pesos de los arcos.

En el caso de que λ_v sea igual a ε_i se procede a eliminar a todos los elementos de la última posición del vector *V*.

Una vez finalizado el documento a procesar, cada nodo del grafo comunicará sus relaciones fuertes al resto para la determinación de los patrones existentes.

```
Siempre que se active v
Si v ∈ V entonces
        ANE(v)
        Si λv = εi , entonces
                Aplicar algoritmo de selección
Sino
        ANGT(v)
```

El siguiente cuadro resume lo explicado en esta sección:

Creencias, Deseos y Plan del nodo *v*		
Creencias	**Deseos**	**Plan**
W_v^m, ε_i, A, $\overline{w}_v^m$, λ_v	$\max\left(\frac{W_v^m}{card(A)}\right)$	Una vez activado v Si $\lambda_v = \varepsilon_i$ Actualizar los pesos de los arcos y crear los arcos nuevos(aplicar ANE(v) y $\Lambda(v)$ Si $\lambda_v \neq \varepsilon_i$ Aplicar algoritmo de selección

CAPITULO 6

Notas al modelo

En este capítulo se hace referencia a particularidades del modelo que deben ser destacadas para comprender de forma integral los procesos descriptos en el mismo. En las páginas que siguen se explicitan aclaraciones, como así también extensiones a lo desarrollado en el capítulo 5.

En la primera sección se describe la relación existente entre el peso de un arco y su posición en el vector Ω*, de esta forma se pretende aclarar situaciones presentes en el proceso de selección de arcos.*

En la segunda sección se explicitan los algoritmos necesarios para el recorrido del grafo y la obtención del listado de patrones que se encuentra representado por el mismo.

En la tercera sección se notifica la posible existencia de ciclos en el grafo y las consecuentes precauciones a tomar para el desarrollo del software.

6.1. Relación de $\overline{w}_i$ y la posición del arco en Ω

En este apartado se hace referencia a la posición de un arco en el vector Ω según su valor $\overline{w}$. En principio se considera sólo el estudio de la posición *q* del mismo debido a que en ella se alojan los arcos de mayor peso, por lo cual, ésta es de mayor relevancia para el momento de la elaboración de los patrones. Luego se generaliza el concepto tratado para todas las posiciones del vector.

Una de las conclusiones arribadas en el capitulo 5 es que todos los arcos que representen una relación fuerte entre gramas se encuentran en la primera posición del vector Ω. Esto permite que la selección de asociaciones frecuentes entre palabras de un documento sea más fácil de localizar y, en consecuencia, más veloz. La característica de posicionamiento de las relaciones más fuertes puede expresarse formalmente mediante el siguiente teorema:

Teorema 6.1.

Sea Ω *el vector que contiene a los arcos salientes del vértice v, q la dimensión* Ω *y*

$$\overline{w}^m{}_i = \frac{w^m(a_i)}{\sum_{a_j \in \bigcup_v^+} w^m(a_j)}$$

Siendo

$w^m(a_j)$ *el peso del arco* a_j *en una instancia m del nodo v.*

$\left[\overline{w}_{\min}; \overline{w}_{\max}\right]$ *el rango de valores posibles para* $\overline{w}_i^m$.

Entonces, si la distribución de los arcos en el vector se realiza según el algoritmo presentado en la sección 5.2, se cumple que

$$\forall a_i \in \bigcup_v^+ \ / \overline{w}_{\min} + (q-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right) < \overline{w}_i^m \rightarrow a_i \in \Omega(q)$$

Dem.:

Para demostrar el teorema anterior se trata de conseguir un absurdo.

Supongamos $\exists a_j \in \bigcup_v^+ / \overline{w}_j^m \geq \overline{w}_{\min} + (q-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right) \wedge a_j \in \Omega(i), i < q$, *entonces*

pueden suscitarse dos situaciones:

1. $\exists t \leq m$ *donde* $a_j \in \Omega(q)$ *desciende de nivel y nunca más se activa*

   Para este caso, a_j *no pertenece a la zona segura de q, por lo tanto,* $\overline{w}_j^t \leq \theta_{seguro}$

   como a_j *nunca más se activa* $\overline{w}_j^m \leq \overline{w}_j^t$ *(absurdo)*

2. $\exists t \leq m$ *donde* $a_j \in \Omega(i), i < q$ *se activa pero no aumenta de nivel, luego* a_j *nunca más se activa.*

   Para este caso, si a_j *no aumenta de nivel, entonces* $\overline{w}_j^t \leq \Omega(i+1)$*, si* a_j *no se activa para otro* $t_n \leq m, t_n \geq t$*, entonces* $\overline{w}_j^m \leq \overline{w}_j^t$ *(absurdo)*

Notar que en el teorema anterior no se pone cota superior a los elementos que se encuentran en el nivel *q* del vector debido a que este nivel posee los arcos más fuertes. Si un arco supera el valor máximo definido por quien implemente el algoritmo, sólo será otro de los arcos fuertes sin que eso cambie el desarrollo normal del algoritmo.

Por otro lado, el recíproco del teorema anterior no es cierto, debido a que, para la determinación del valor de ε se supone una espera máxima de tiempo posible para descender de nivel a un arco (*ver sección 5.2.3*), puede suceder que un arco que deba descender de posición en un tiempo $\varepsilon_h < \varepsilon$ no lo haga en ese momento por lo que permanecerá en una posición incorrecta durante un tiempo igual a $\varepsilon - \varepsilon_h$. Esto no es contraproducente para el algoritmo salvo que el análisis de texto se complete antes del tiempo ε. Si una vez culminado el proceso de selección de arcos se procede a la conformación de los patrones con los arcos que se encuentran en la primera posición del vector Ω pueden utilizarse arcos no lo suficientemente fuertes para dicha acción. Esto indica que durante el recorrido del grafo para la elaboración del listado de patrones se debe realizar la corroboración de los pesos de los arcos del primer nivel de Ω.

La figura 6.1 muestra un posible caso que invalida el reciproco del teorema 6.1.

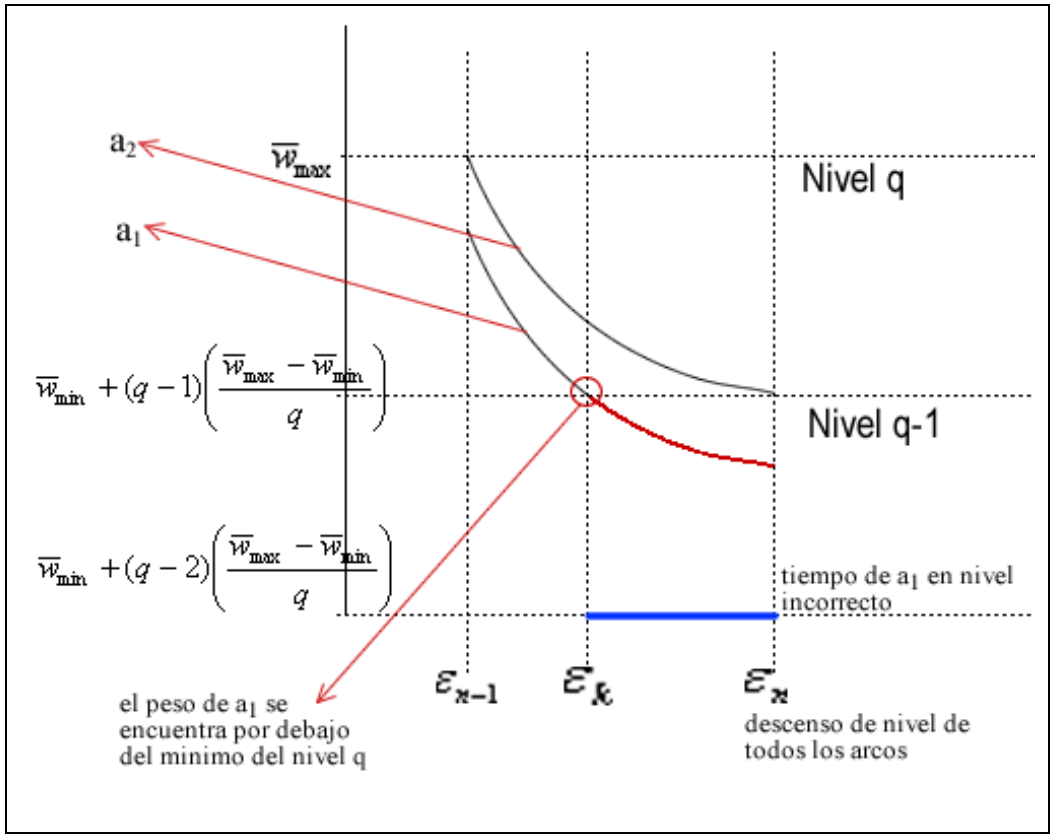


Figura 6.1. Cambio de nivel de arcos

Finalmente, se puede observar que lo determinado hasta el momento sólo refiere a particularidades que se pueden observar a partir de la aplicación del algoritmo determinado en la sección 5.2, sin embargo, lo explicitado anteriormente sólo puntualiza lo que ocurre en la posición *q* del vector Ω. Las distinciones realizadas pueden generalizarse para todo arco dispuesto en cualquier posición del vector.

Teorema 6.2.

Sea Ω *el vector que contiene a los arcos salientes del vértice v, q la dimensión* Ω *y*

$$\overline{w}^{m}{}_{i} = \frac{w^{m}(a_i)}{\sum_{a_j \in \bigcup_v^+} w^{m}(a_j)}$$

Siendo

$w^{m}(a_j)$ *el peso del arco* a_j *en una instancia m del nodo v.*

$[\overline{w}_{\min}; \overline{w}_{\max}]$ *el rango de valores posibles para* $\overline{w}_i^{m}$.

Entonces, si la distribución de los arcos en el vector se realiza según el algoritmo presentado en la sección 5.2, se cumple que

$$\forall a_j \in \bigcup_v^+ \ / \overline{w}_{\min} + (i-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right) < \overline{w}_j^m \rightarrow a_j \in \Omega(i)$$

Dem.:

La demostración para este caso es análoga a la desarrollada para el teorema 6.1

Supongamos $\exists a_j \in \bigcup_v^+ / \overline{w}_j^m \geq \overline{w}_{\min} + (i-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right) \wedge a_j \in \Omega(l), l < i$ *, entonces pueden suscitarse dos situaciones:*

3. $\exists t \leq m$ *donde* $a_j \in \Omega(i)$ *desciende de nivel y nunca más se activa*

   Para este caso, a_j *no pertenece a la zona segura de q, por lo tanto,* $\overline{w}_j^t \leq \theta_{seguro}$

   como a_j *nunca más se activa* $\overline{w}_j^m \leq \overline{w}_j^t$ *(absurdo)*

4. $\exists t \leq m$ *donde* $a_j \in \Omega(l), l < i$ *se activa pero no aumenta de nivel, luego* a_j *nunca más se activa.*

   Para este caso, si a_j *no aumenta de nivel, entonces* $\overline{w}_j^t \leq \Omega(l+1)$ *, si* a_j *no se activa para otro* $t_n \leq m, t_n \geq t$ *, entonces* $\overline{w}_j^m \leq \overline{w}_j^t$ *(absurdo)*

6.2. Recorrido del grafo y construcción de patrones

Uno de los temas que resta tratar en cuanto a la resolución de los patrones es el proceso por el cual se recorre el grafo final y se recuperan las asociaciones detectadas. La búsqueda de patrones en el grafo se desarrolla según la visión de cada uno de los agentes que se encuentran inmersos en el mismo. El proceso de construcción de patrones se desarrolla en tres etapas: construcción de patrones locales, publicación de patrones y recorrido de nodos. Estas instancias se detallan a continuación.

6.2. a. Construcción de patrones locales

Una vez finalizado el análisis del texto y el algoritmo de clasificación de arcos (*ver sección 5.2*) cada uno de los vértices conoce sus arcos salientes más relevantes, es decir los de mayor peso. Sin embargo, cada uno de los agentes no conoce los arcos relevantes de su nodo adyacente, por lo tanto, como primera instancia en la construcción de patrones un agente sólo puede elaborar los patrones de forma local.

La construcción de los patrones de forma local consiste en la concatenación de los gramas que representan los nodos adyacentes del grafo. Cada agente recorre su lista de arcos fuertes y solicita a su vértice vecino se le comunique el grama que representa. De esta forma, el agente adquiere el conocimiento necesario para desarrollar esta tarea. El algoritmo de elaboración de patrones locales se detalla a continuación:

Algoritmo 6.1: Construcción de patrones locales.

Entrada: α_s , $\sum$ *// Separador de patrón y alfabeto*

$\Phi = 0$

Si $\alpha_s \in W(\sum)$, entonces

  Terminar algoritmo

  Salida Φ

Fin Si

$\forall a_j \in \bigcup_v^+ \ / a_j \in \Omega(q)$, hacer

  Si $\overline{w}_j^m > \overline{w}_{\min} + (q-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right)$, entonces *// sección 6.1*

    α =Grama del agente vecino a_{jv2}

    $\Phi = \Phi \bigcup \{v_\alpha . \alpha_s . \alpha\}$

  Fin Si

Fin

Salida: Φ

El desarrollo puede verse más claramente en la figura 6.2.

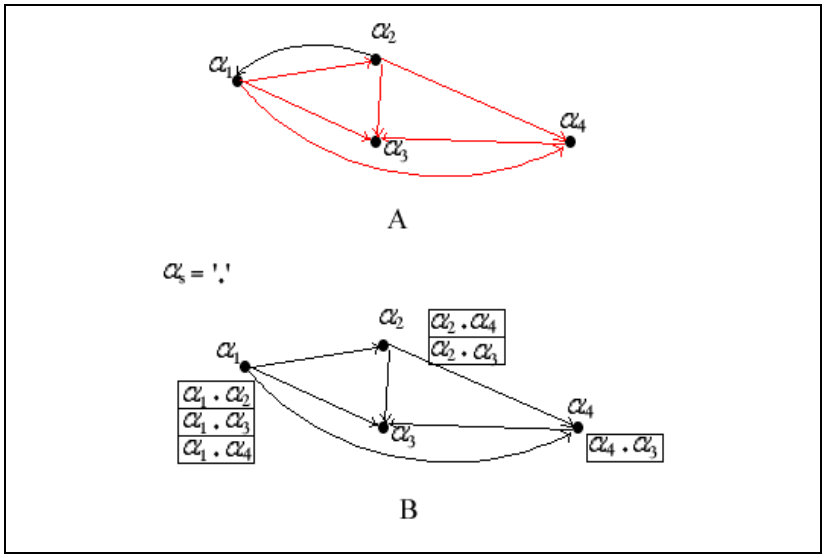


Figura 6.2. Proceso de construcción de patrones locales.
Se omite el valor de los pesos de los arcos.
A) Arcos relevantes en rojo
B) Construcción del conjunto Φ

6.2. b. Construcción de patrones globales

El algoritmo de construcción de patrones globales consiste, básicamente, en un proceso iterativo donde cada uno de los agentes del grafo comunica a su vecino los patrones que conoce. El algoritmo presenta la particularidad que en cada iteración, cada agente registra nuevos patrones y, cada agente termina el proceso de construcción de patrones cuando no aprende más.

El proceso de construcción de patrones posee 3 etapas:

1. Construcción de Φ_0 : En la cual el agente crea el conjunto Φ_0 donde contiene los gramas de los agentes vecinos.
2. Publicación de Patrones: Donde cada agente comunica a su vecino los patrones que conoce en esa iteración. El agente que recibe la información la procesa para crear nuevos patrones. Cada publicación de los patrones implica una nueva iteración en el algoritmo.
3. Construcción de patrones globales: El agente, una vez que culmina el proceso de publicación, construye los patrones globales que él conoce.

A continuación se muestran los algoritmos mencionados.

Algoritmo 6.2: Construcción de Φ_0

$\Phi_0 = 0$

$\forall a_j \in \bigcup_v^+ \ / \ a_j \in \Omega(q)$, hacer

Si $\overline{w}_j^m > \overline{w}_{\min} + (q-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right)$, entonces *// sección 6.1*

α =Grama del agente vecino a_{jv2}

$\Phi_0 = \Phi_0 \bigcup \{\alpha\}$

Fin Si

Fin

Salida: Φ_0

Algoritmo 6.3: Publicación de patrones

Entrada: α_s , Φ_0 , Φ_1

$\Phi'_1 = \Phi_1$

$\forall a_j \in \bigcup_v^+ \ / \ a_j \in \Omega(q)$, hacer

Si $\overline{w}_j^m > \overline{w}_{\min} + (q-1)\left(\frac{\overline{w}_{\max} - \overline{w}_{\min}}{q}\right)$, entonces *// sección 6.1*

Φ =Obtener patrones de a_{jv2}

$\sum_0 = \Phi_0 \bigcup \{\alpha_s\}$

$\mathrm{A} = \Phi \bigcap W(\sum_0)$

Fin Si

Fin

$\Phi_1 = \Phi_0 \bigcup \mathrm{A}$

Si $\Phi_1 - \Phi'_1 \neq 0$, entonces

Publicación de Patrones $(\alpha_s, \Phi_0, \Phi_1)$

Fin Si

Algoritmo 6.4: Construcción de patrones globales

Entrada: α_s , $\sum$ *// Separador de patrón y alfabeto*

Si $\alpha_s \in W(\sum)$, entonces
  Terminar algoritmo
  Salida por error
Fin Si

$\Phi_0 =$ Construcción de Φ_0

$\Phi_1 = 0$

Publicación de Patrones $(\alpha_s, \Phi_0, \Phi_1)$

$\Phi_{Patrones\_v} = \{v_\alpha . \alpha_s\} . \Phi_1$

Salida: $\Phi_{Patrones\_v}$

Veamos ahora el proceso de construcción de patrones globales continuando el ejemplo de la sección anterior:

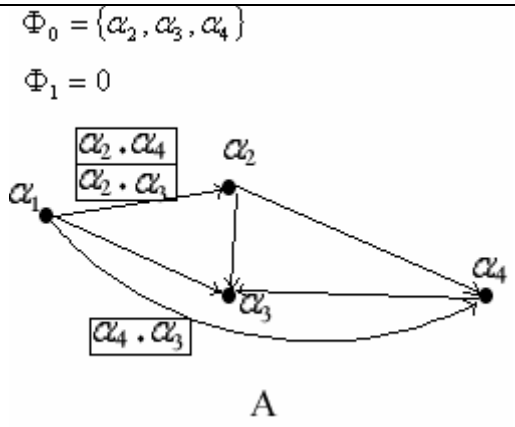


Figura 6.3.A. En la primera etapa los nodos vecinos comunican sus patrones locales.

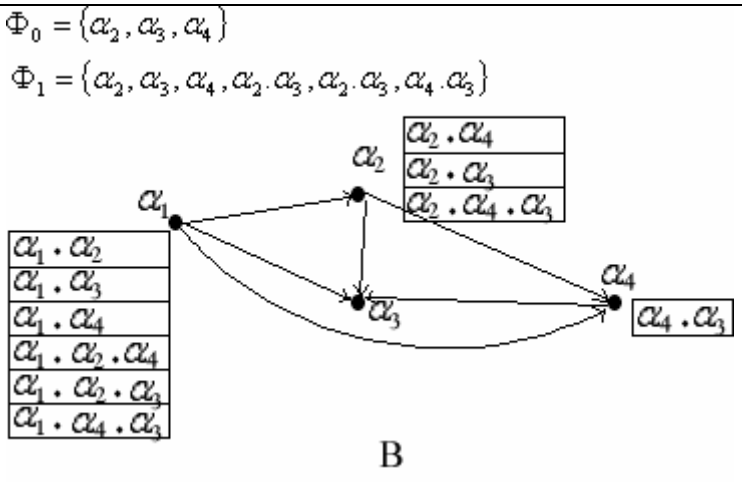


Figura 6.3.B. El agente recopila la información recibida y construye los primeros patrones globales.

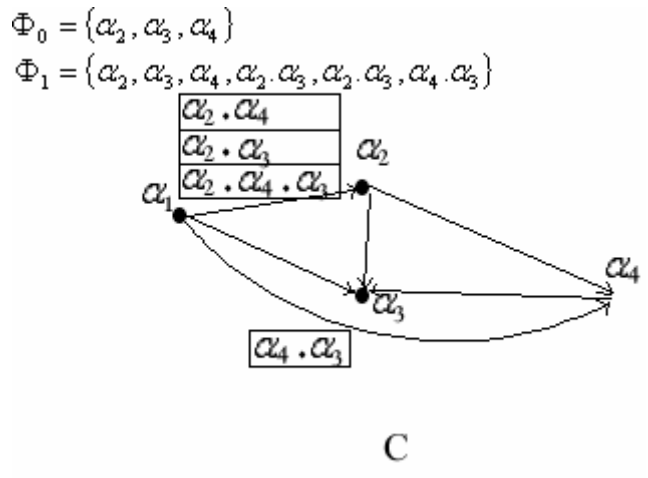


Figura 6.3.C. Los agentes vuelven a comunicarse sus patrones.

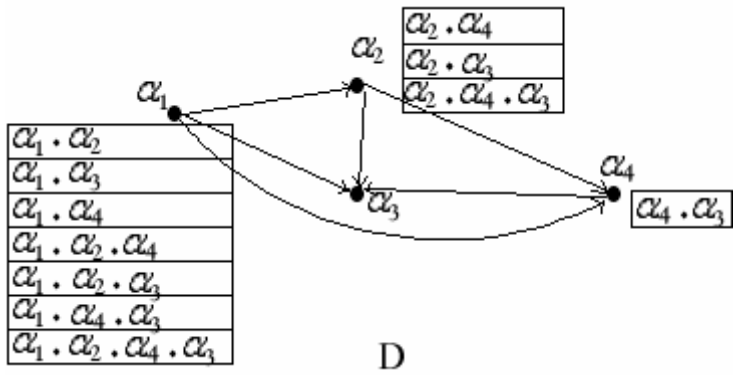


Figura 6.3.C. Los agentes procesan la última información recibida para formar todos los patrones globales.

6.2. c. Recorrido de nodos

El recorrido de nodos es el proceso más simple que se desarrolla en la etapa final. Fundamentalmente consiste en la recolección de los patrones globales de cada agente y la elaboración del listado final de patrones. Este listado puede ser referido hacia cualquier otro sistema que lo procese, debe quedar en claro que lo resuelto por el método que aquí se plantea es la detección de patrones de textos, la finalidad que se le de a este resultado depende de las motivaciones de quién implemente el algoritmo y los objetivos que desea cumplir.

Algoritmo 6.5. Recorrido de nodos

Sea $H = (V, E)$,

$\Phi_{Final} = 0$

$\forall v \in V$
    $\Phi_{Final} = \Phi_{Final} \bigcup$ Solicitar Patrones a v
Fin

Salida: Φ_{Final}

6.3. Control de ciclos en el grafo

Debido al carácter genérico del algoritmo, la estructura del texto que se desee procesar es totalmente aleatoria. Esto deriva en que el grafo resultante del proceso pueda tener ciclos, o no. Un ciclo en el grafo conlleva a una recursión infinita en los algoritmos antes desarrollados. Para evitar esta interrupción en el programa se debe controlar la longitud de los patrones detectados. Es decir, el algoritmo de detección de publicación de patrones (*algoritmo 6.3*) es conveniente detenerlo cuando no se presente alguna modificación en el listado obtenido y cuando los patrones detectados en la última iteración superen la longitud máxima definida por quien implemente el algoritmo.

CAPITULO 7

Resultados Experimentales

En el presente capitulo se evalúa, en función de la herramienta desarrollada, las características del modelo presentado en esta tesis.

En principio se realiza un estudio del impacto de la implementación de agentes al modelo de grafos para la detección de patrones en textos sobre la performance en tiempo.

En la segunda sección se estima el orden del algoritmo presentado.

En la tercera sección se analizan los resultados de la clasificación de textos realizada utilizando lo explicado a lo largo de este documento.

7.1. Estudio de la Influencia del uso de Agentes en el tiempo de ejecución

En el primer estudio realizado se analiza la influencia de la aplicación de la teoría de agentes al grafo para la detección de patrones en textos en cuanto al factor performance de tiempo. El procedimiento realizado para este estudio consiste en la ejecución de la herramienta dos veces:

1. para la primera ejecución se desactivan los agentes y todas sus funcionalidades, es decir, no existe selección de arcos mientras se procesa el texto y, en consecuencia, no hay eliminación de los mismos. Para este caso, la detección de relaciones relevantes en el grafo y la conformación de patrones se realiza una vez obtenido el grafo final.

2. para la segunda corrida se utiliza el modelo explicado en los capítulos anteriores. Los parámetros utilizados fueron:

- Tamaño de Ventana de Visualización: 10
- Dimensión del vector Ω : 3
- $\overline{w}_{\min} = 0.03$
- $\overline{w}_{\max} = 0.24$
- $\xi(d) = \frac{1}{x+1}$

El elemento que se utilizó como variable es la cantidad de palabras. Las mediciones se realizaron sólo sobre la ejecución del modelo, no sobre las acciones de parseo del texto. Es necesario destacar que, debido a que este es un análisis de tiempos, se tomó en consideración a todas las palabras del texto. Es decir, no se omitió ninguna para detección de patrones, sin importar su conjugación o frecuencia en el idioma.

Los resultados obtenidos son:

Cantidad de palabras en el documento	***Sin Agente (seg)***	***Agente (seg)***	***Variación***
2000	0.134	0.101	% 25
4000	1.004	0.324	% 68
8000	1.670	0.621	% 40
16000	2.233	1.152	% 49
32000	3.421	1.349	% 61
64000	7.021	2.244	% 68
128000	13.12	5.127	% 61
256000	28.31	10.131	% 64
512000	59.42	23.12	% 61
1024000	----	49	

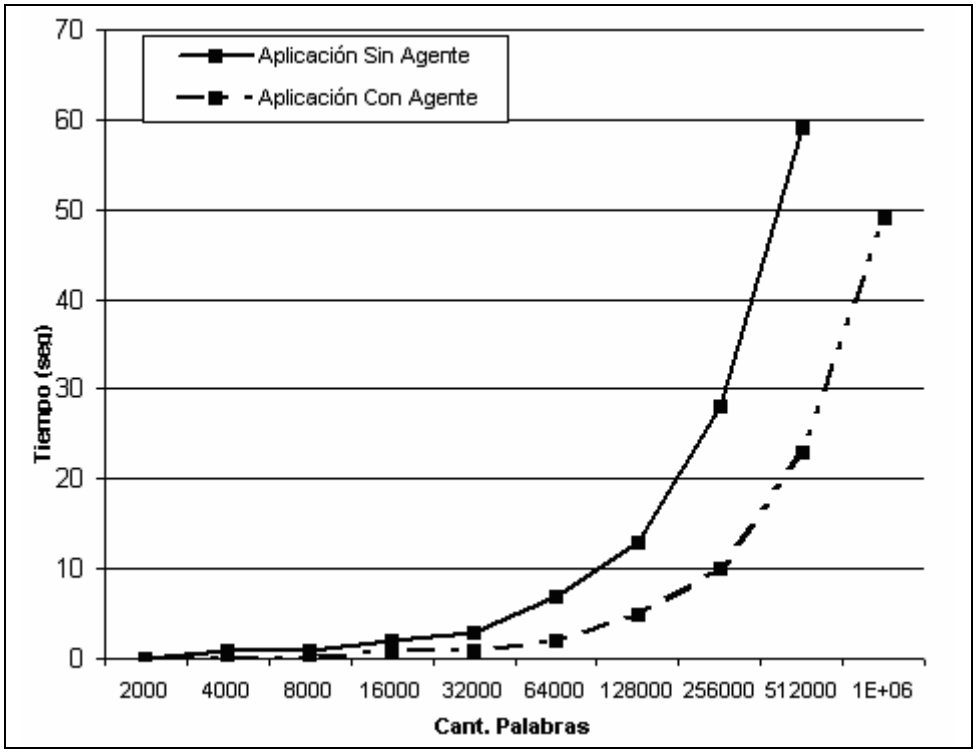


Figura 7.1. Comparación de la ejecución del modelo con y sin la aplicación de Agentes.
Fuente de Textos: Project Gutenberg. http://www.gutenberg.org/. Idioma: Inglés.

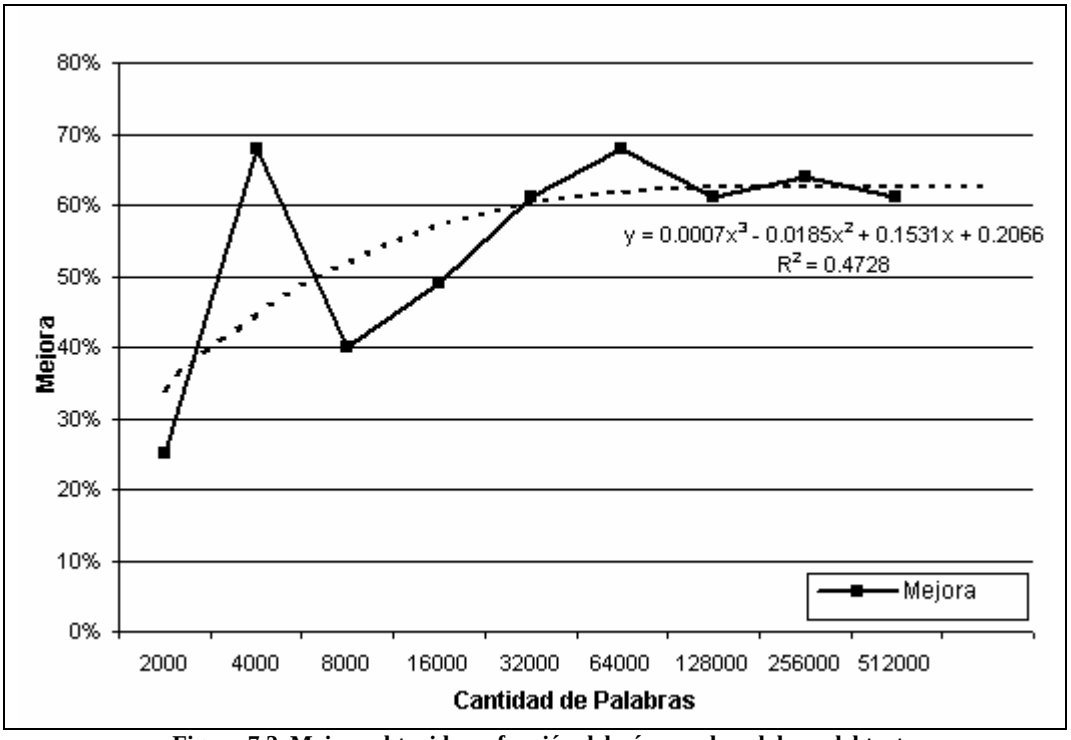


Figura 7.2. Mejora obtenida en función del número de palabras del texto.
Fuente de Textos: Project Gutenberg. http://www.gutenberg.org/. Idioma: Inglés.

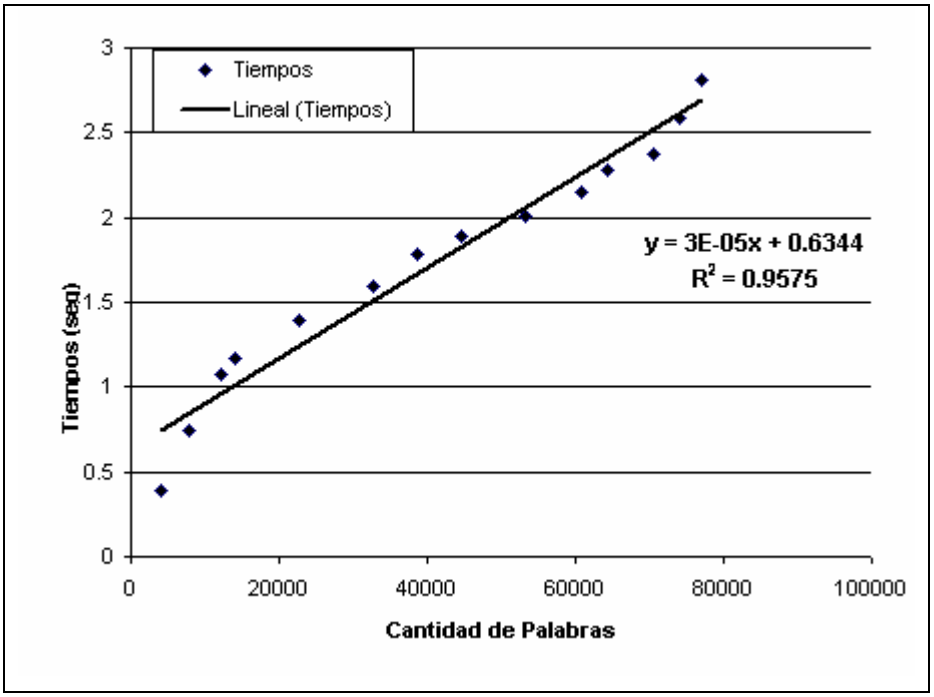
Observando los resultados obtenidos se puede remarcar que la incorporación de los agentes al modelo implica un mejora de tiempo de aproximadamente un 60%. La mejora en tiempo tiene carácter polinomial con una asíntota en el 63%. Por otra parte, el análisis de textos con más de un millón de palabras es totalmente inviable si no se considera un método de reducción del volumen del grafo.

7.2. Estimación del orden del algoritmo

Para la estimación experimental del orden del algoritmo se propone la ejecución del modelo con una variación proporcional de la longitud de textos analizados, el elemento medido es el tiempo.

Los resultados obtenidos son:

Cantidad de Palabras	Tiempos (seg.)
4131	0.39375
7811	0.746875
12102	1.075
14149	1.16875
22602	1.39775
32776	1.591875
38635	1.780375
44521	1.888524
53154	2.007855
60842	2.145421
64223	2.27775
70461	2.3711
74152	2.58141
77146	2.8145



7.3. Estudio de la calidad de los patrones obtenidos

La herramienta desarrollada utiliza el modelo descrito para la clasificación automática de textos en función de los patrones detectados en ellos. Para cumplir con este objetivo el sistema requiere un conjunto de textos clasificados previamente para establecer una base de conocimiento y luego poder clasificar textos futuros. Estableciendo la capacidad de predicción de la herramienta se puede estimar la calidad y representatividad de los patrones detectados en los textos.

Para desarrollar esta experiencia se utilizaron 20000 mensajes del grupo Usenet. Este conjunto de textos puede ser descargado del sitio http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html . Esta información se encuentra clasificada de la siguiente manera:

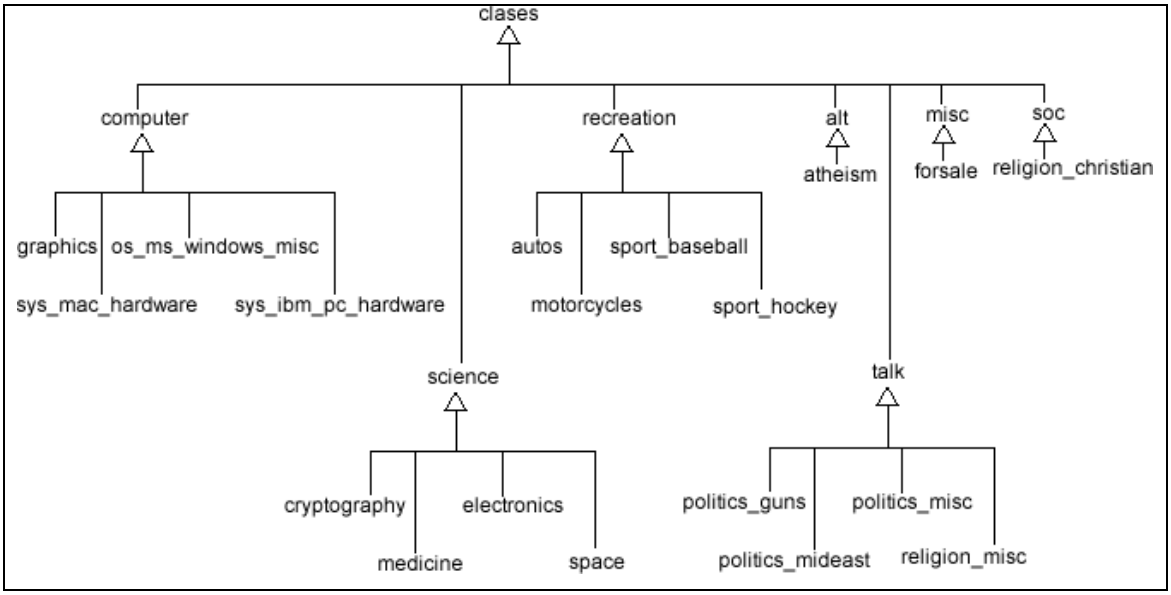


Figura 7.4. Clases de textos

Se puede observar que los textos que pertenecen a una misma superclase se encuentran relacionados, sin embargo, existen categorías de textos que, si bien, no pertenecen a una misma superclase se puede prever que se encuentran relacionadas entre sí. Es por ello que para el análisis de los resultados obtenidos se consideró también la similitud de los temas tratados por cada una de las clases. En las siguientes imágenes se representan aquellas clases que se encuentran vinculadas por un tema.

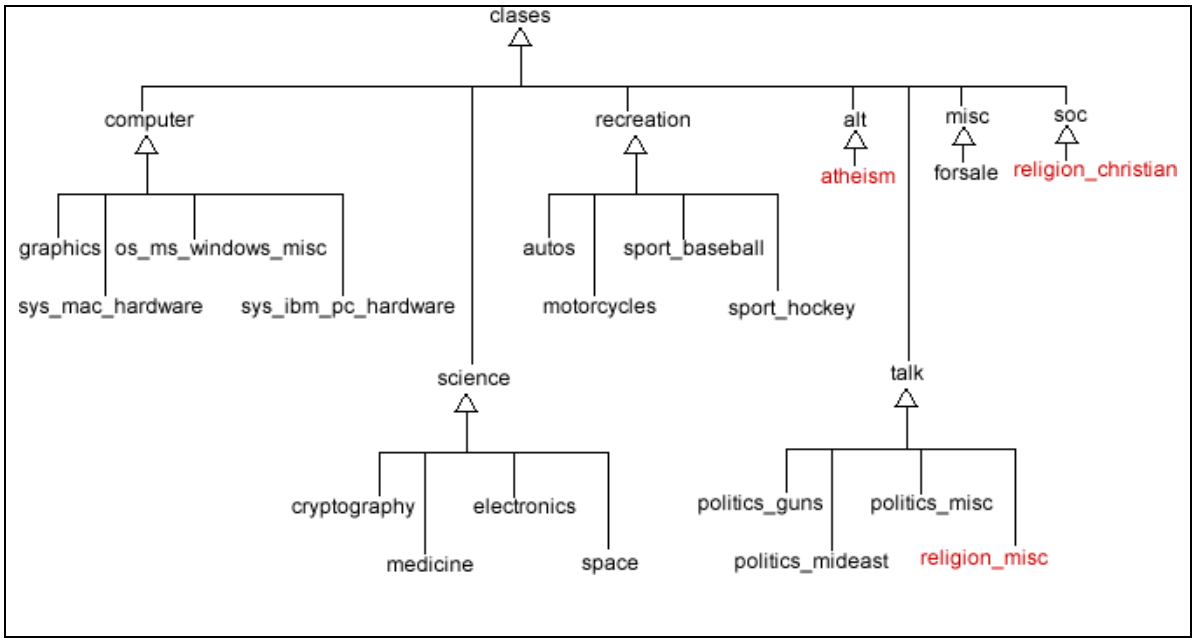


Figura 7.5. Clases de textos relacionados por tema religioso

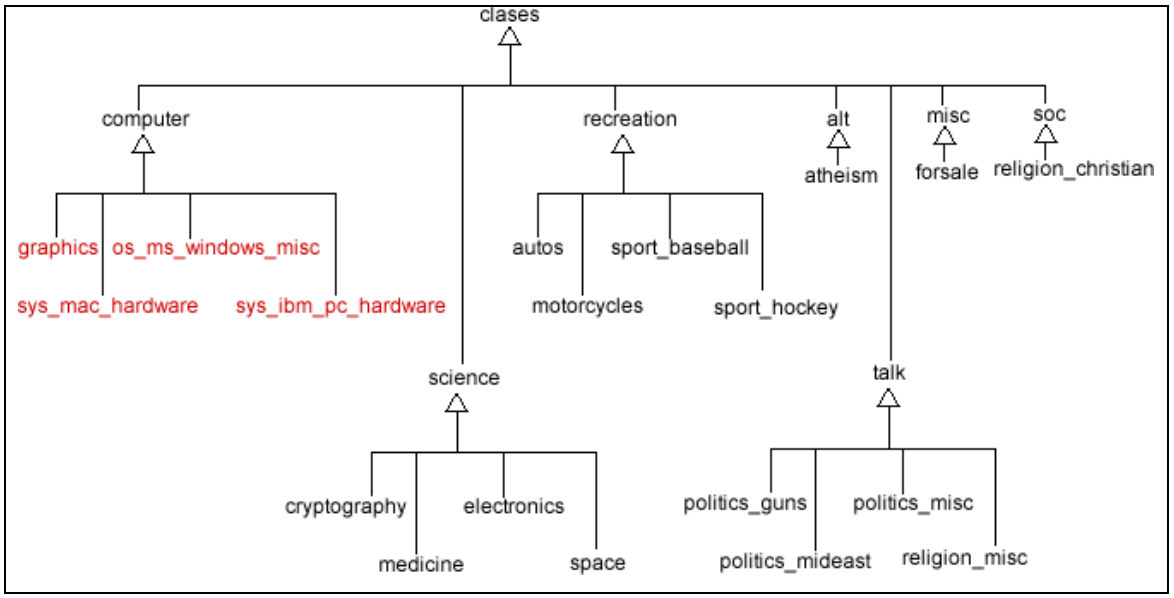


Figura 7.5. Clases de textos relacionados por tema computer

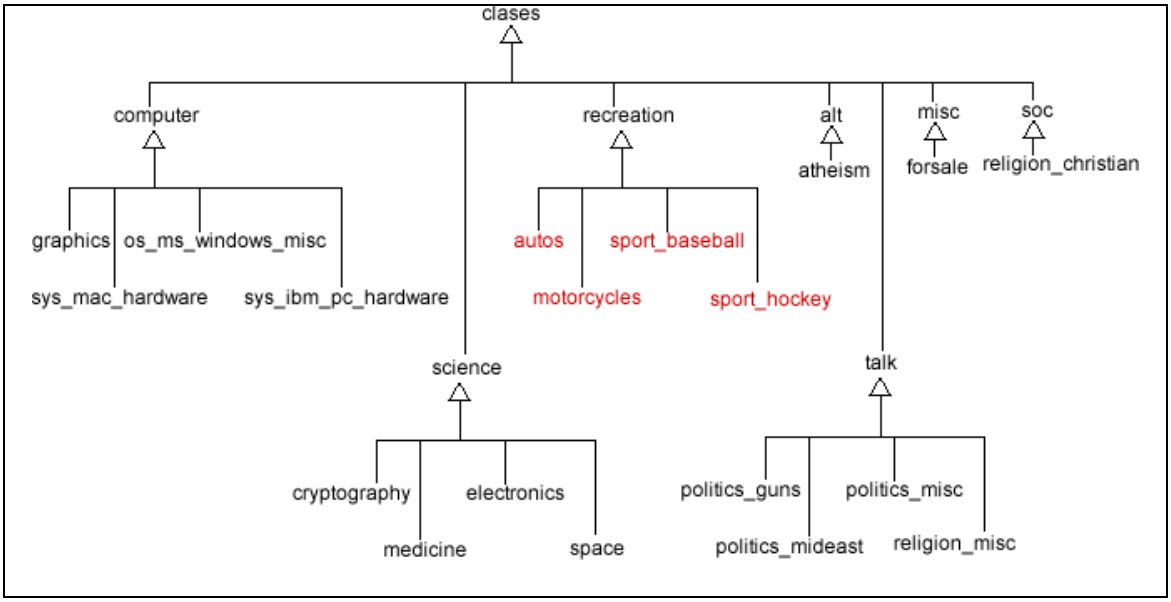


Figura 7.6. Clases de textos relacionados por tema recreation

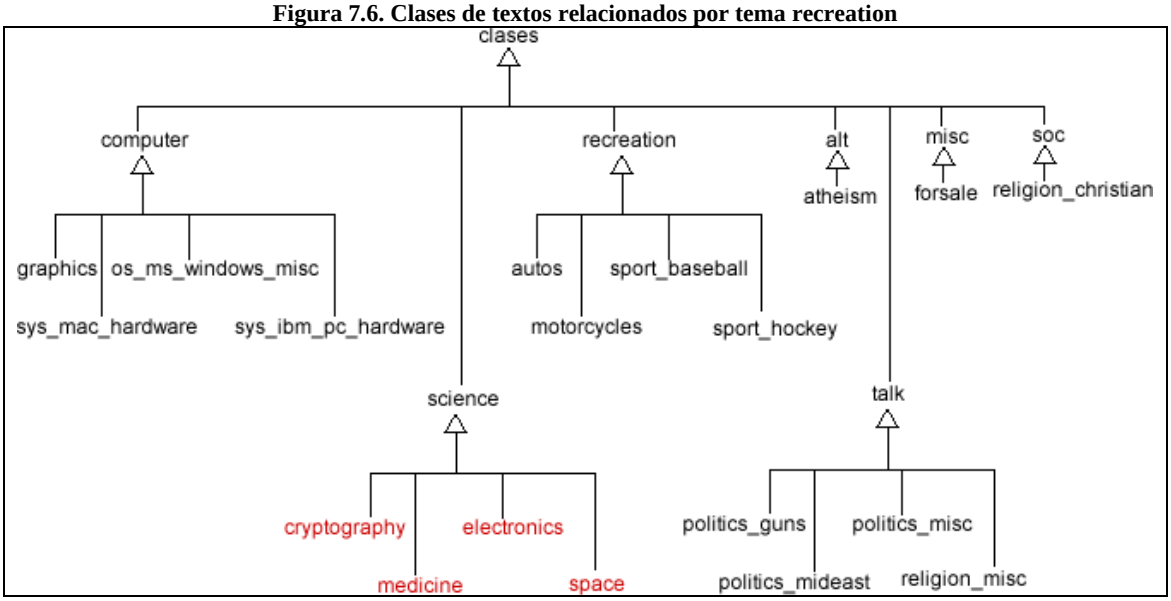


Figura 7.7. Clases de textos relacionados por tema science

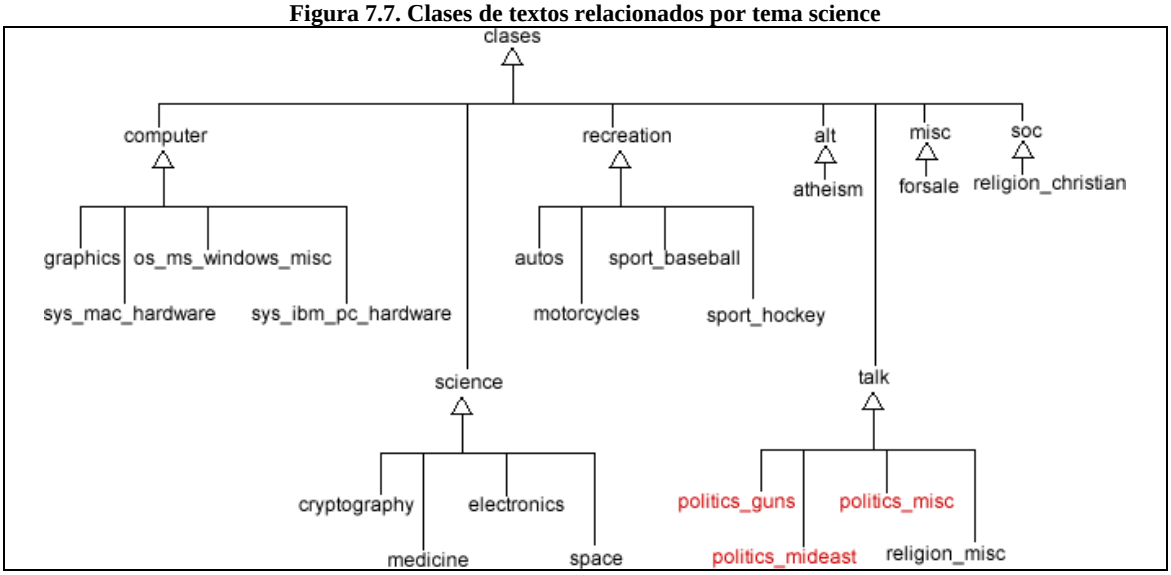


Figura 7.8. Clases de textos relacionados por tema politics.

Para el entrenamiento del sistema se utilizó el 65% de los registros, asimismo el 35 % restante fue utilizado para las pruebas del sistema. Los parámetros especificados fueron:

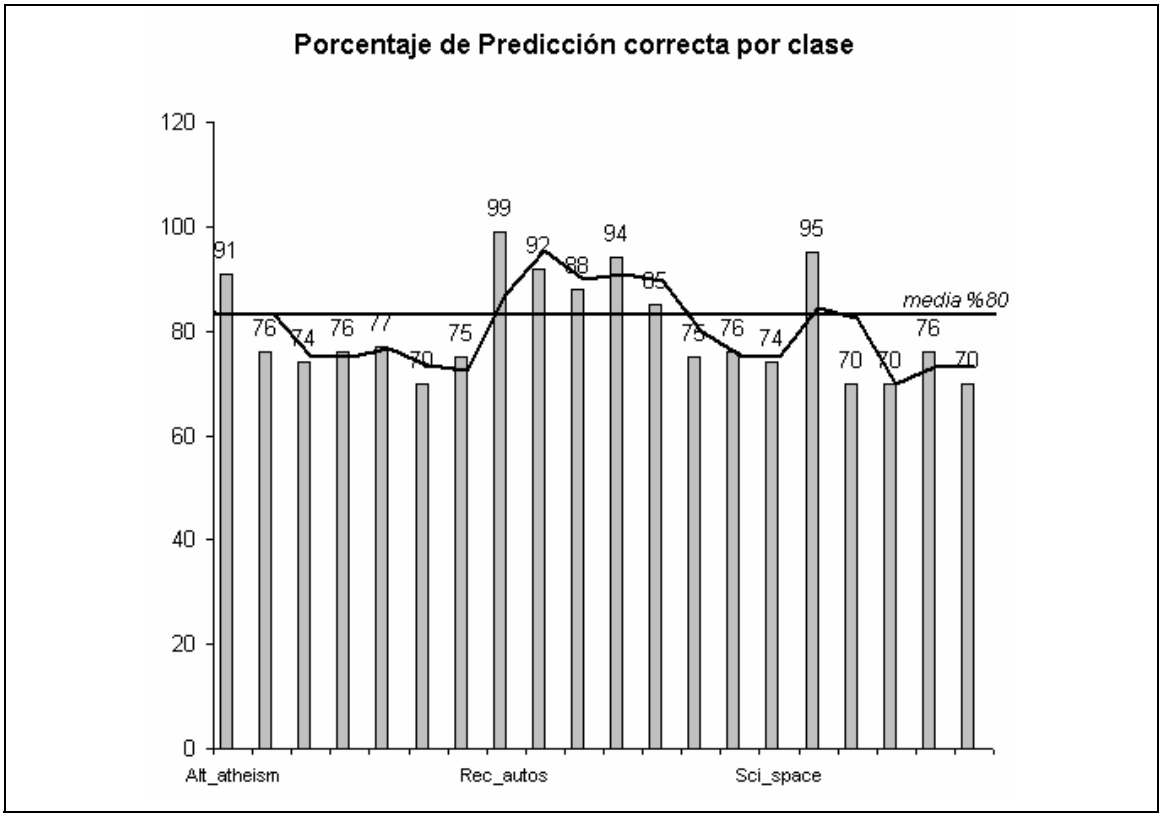
Entrenamiento	Evaluación
• Tamaño de Ventana de Visualización: 10 • Dimensión del vector Ω : 3 • $\overline{w}_{\min} = 0.05$ • $\overline{w}_{\max} = 0.15$ • $\xi(d) = e^{-x}$	• Tamaño de Ventana de Visualización: 10 • Dimensión del vector Ω : 3 • $\overline{w}_{\min} = 0.01$ • $\overline{w}_{\max} = 0.08$ • $\xi(d) = e^{-x}$

NOTA: Los parámetros especificados tanto para el entrenamiento como para la evaluación, se estimaron a partir del análisis de una muestra reducida de los textos a procesar. Se realizaron una serie de corridas previas con un número reducido de textos para el ajuste de los parámetros. En principio, se determinó la función $\xi(d)$ con el fin de ponderar fuertemente las distancias pequeñas entre gramas y descartar aquellas relaciones entre gramas alejados entre sí. Por otro lado, para el tamaño de la ventana de visualización se consideró, en principio, la cantidad promedio de gramas en las oraciones de los textos a procesar. Finalmente, estos parámetros como así también la dimensión del vector Ω y los pesos máximos y mínimos se ajustaron empíricamente en función de las ejecuciones del software antes mencionadas.

Clase	% Aciertos	% No Clasificados	% Clase similar	% Clase sin similitud	Cantidad de registros
Alt_atheism	91	44	50	6	400
Comp_graphics	76	44	38	18	396
Comp_os_ms_windows_misc	74	48	32	20	349
Comp_sys_ibm_pc_hardware	76	42	40	18	389
Comp_sys_mac_hardware	77	55	35	10	367
Comp_windows_x	70	33	48	19	341
misc_forsale	75	46	0	54	337
Rec_autos	99	100	0	0	371
Rec_motorcycles	92	39	18	43	348
Rec_sport_baseball	88	35	23	42	368
Rec_sport_hockey	94	36	24	40	370
Sci_crypt	85	55	15	30	366
Sci_electronics	75	41	22	37	307
Sci_med	76	38	15	47	376
Sci_space	74	43	20	37	320
Soc_religion_christian	95	35	50	15	322
Talk_politics_guns	70	64	20	16	329
Talk_politics_mideast	70	74	15	11	357
Talk_politics_misc	76	60	23	17	357
Talk_religion_misc	70	30	63	7	329

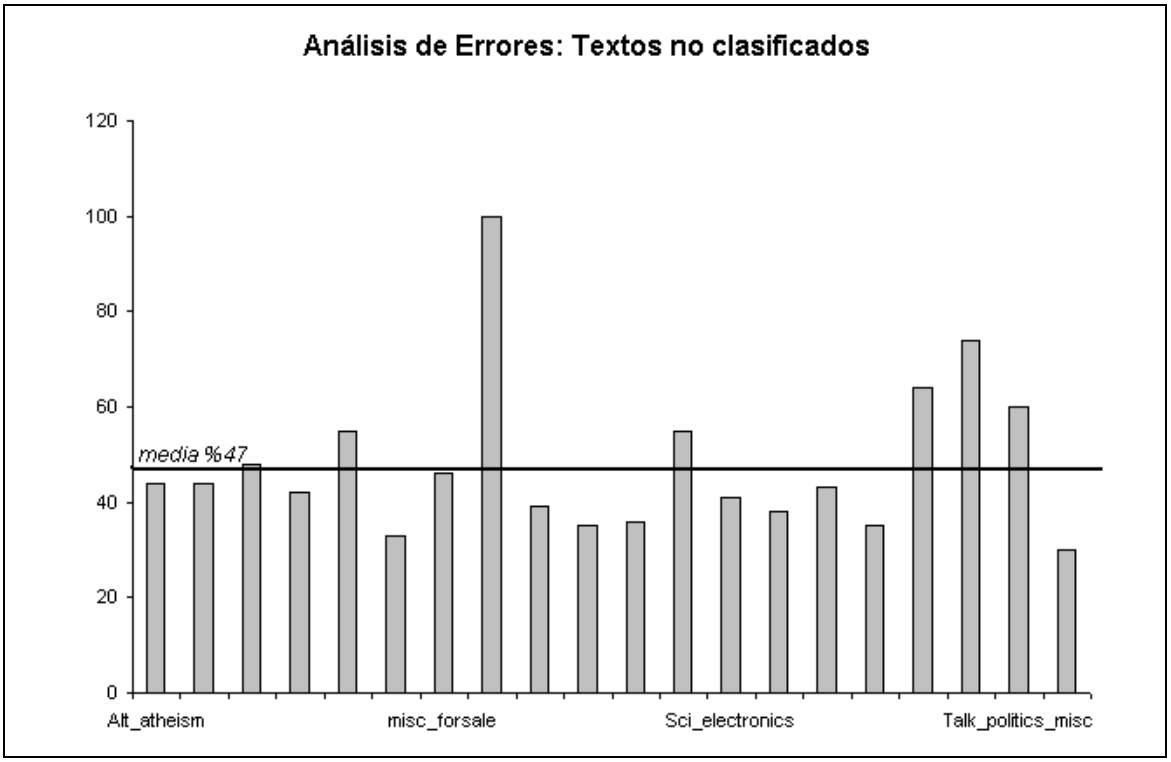
7.3. a. Estadísticas por clase

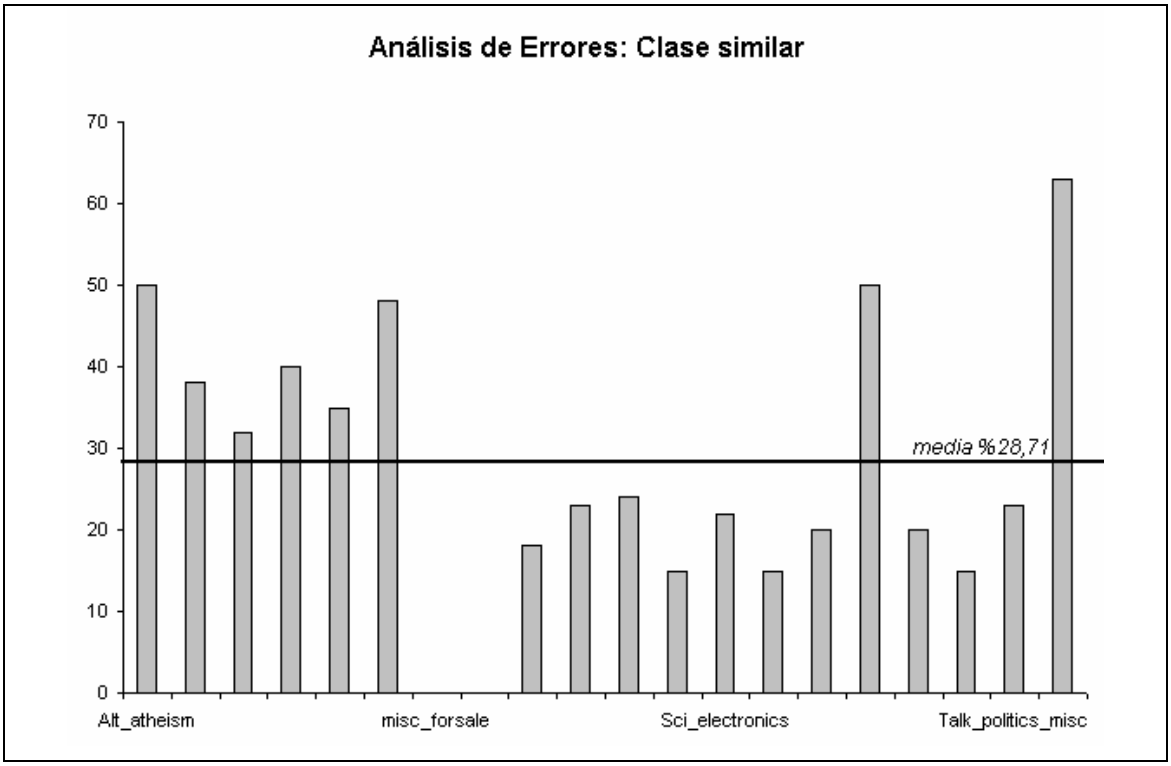
Las siguientes gráficas representan las estadísticas realizadas para el análisis de la capacidad predictiva de la herramienta según la clase.

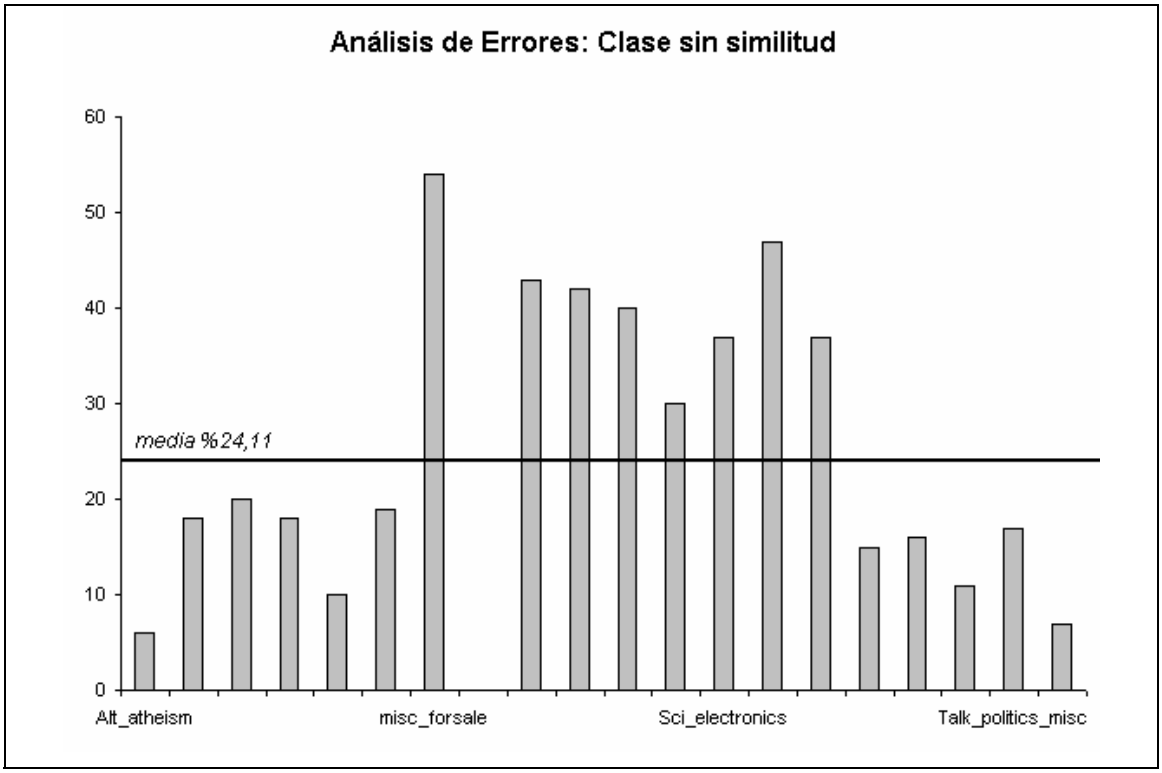


Para el análisis de los errores de clasificación de la herramienta se toman los porcentajes en relación al número total de textos, por clase, clasificados incorrectamente. Un incorrectamente clasificado puede ser:

1. **No Clasificado**: La herramienta no asocia al texto con ninguna clase.
2. **Clase Similar**: La herramienta asocia al texto con una clase similar a la que realmente pertenece. (*sección 7.3*)
3. **Clase sin Similitud**: La herramienta asocia al texto con una clase sin similitud.

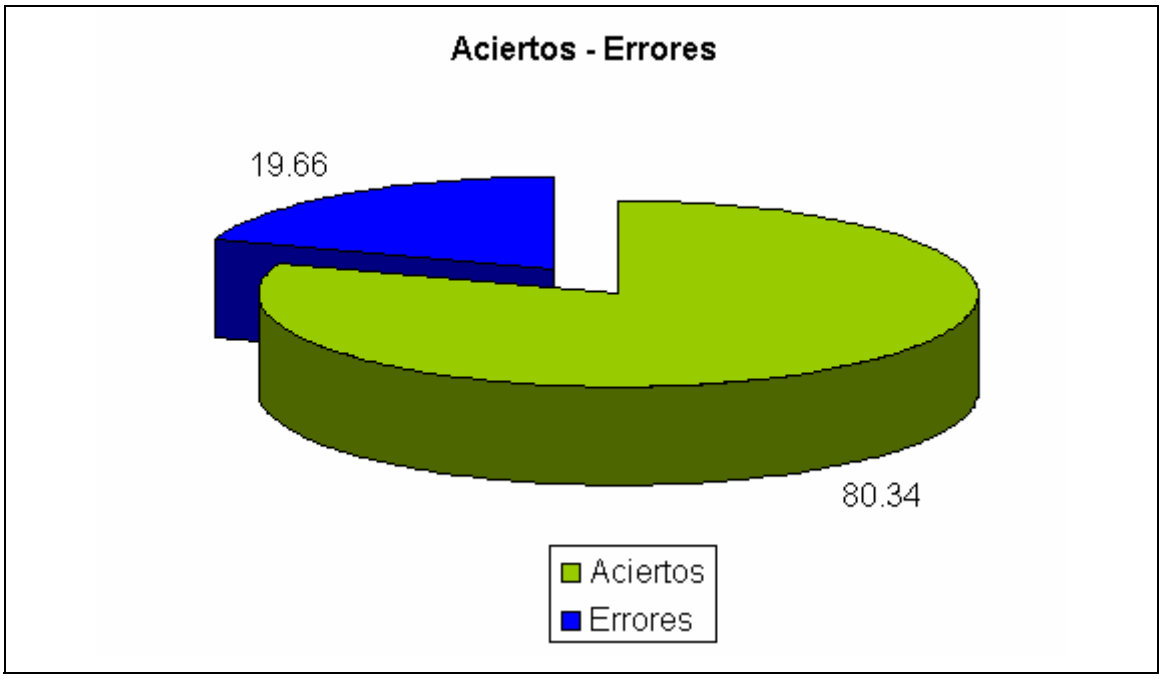

Análisis de Errores: Textos no clasificados
120
100
80
60
40
20
0
media %47
Alt_atheism
misc_forsale
Sci_electronics
Talk_politics_misc


Análisis de Errores: Clase similar
70
60
50
40
30
20
10
0
media %28,71
Alt_atheism
misc_forsale
Sci_electronics
Talk_politics_misc

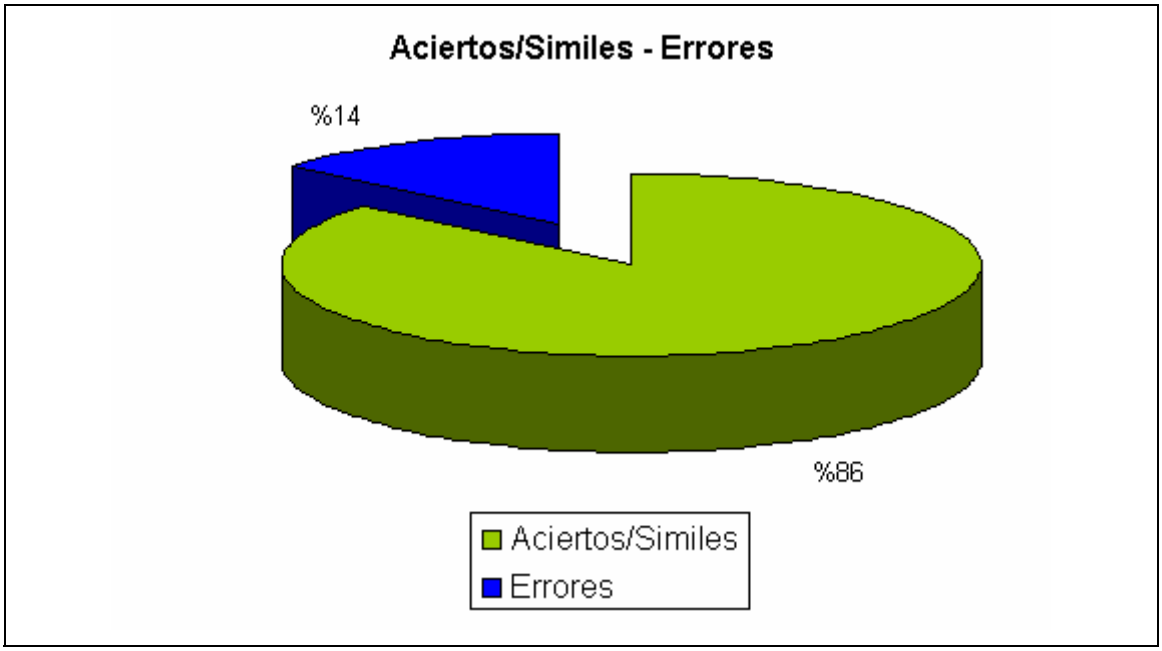

Análisis de Errores: Clase sin similitud
60
50
40
30
20
10
0
media %24,11
Alt_atheism
misc_forsale
Sci_electronics
Talk_politics_misc

7.3. b. Resumen

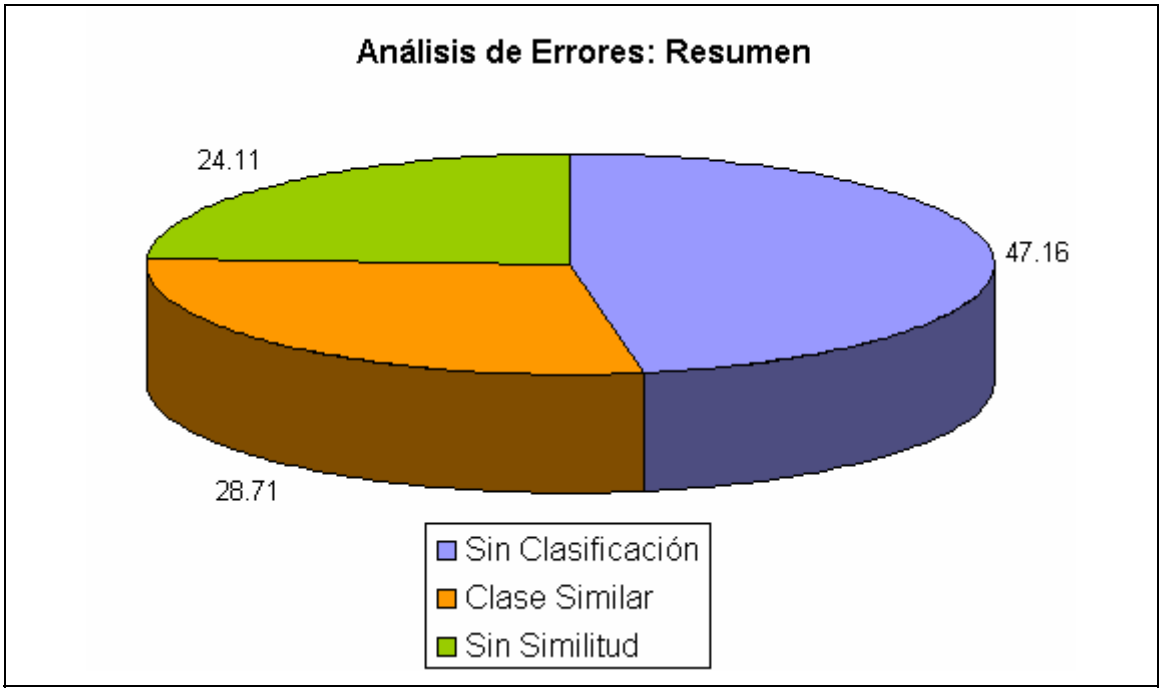
Las gráficas siguientes sintetizan los resultados obtenidos en la ejecución de la herramienta. En principio se esquematiza el porcentaje de aciertos de clasificación sobre el total de registros, se puede observar que la herramienta predice correctamente en aproximadamente un 80% de los casos evaluados.



Por otro lado, si consideramos como acierto a la vinculación de un texto con una clase similar a la que realmente pertenece es porcentaje de aciertos asciende a 86%.



Finalmente, si se analizan los errores de predicción cometidos se observa que el 24% de los textos incorrectamente clasificados se vinculan con una clase sin similitud a la clase que realmente perteneces, el 47% no se asocia a ninguna clase y el 29% se identifica con una clase similar a la real.



CAPITULO 8

Conclusión y Trabajos Futuros

El método propuesto para detección de patrones en textos permite al usuario definir el tipo de patrón que desea buscar mediante la determinación analítica de la función peso. Esto hace al algoritmo altamente flexible debido a que se pueden detectar patrones en distintos textos sin importar cuestiones como por ejemplo la longitud o el idioma. Es así que, el modelo presentado en esta tesis es independiente de la aplicación del mismo, dejando a quien lo implemente la posibilidad de adaptarlo mediante la determinación de los diversos parámetros.

Asimismo, la inclusión de la teoría de agentes y el método de selección presentado permite reducir el volumen del grafo disminuyendo los tiempos de procesamiento y aumentando la especificidad de los patrones detectados. Esta cualidad permite que el modelo sea implementado en programas donde se requiera un tiempo de procesamiento corto, como por ejemplo, un explorador de Internet para la clasificación de las páginas Web que muestre. Observando los resultados de las pruebas realizadas se tiene que la mejora en cuanto al tiempo del sistema con la inclusión de los agentes es de aproximadamente 60%.

Por otro lado, si se analizan los resultados experimentales se observa que la calidad predictiva de la herramienta desarrollada para la prueba del modelo es de alrededor de un 86% lo que representa que los patrones detectados por el mismo son representativos de la clase de texto que se procese. Es decir, la calidad de los patrones es significativa, no obstante este parámetro de evaluación puede ser mejorado o empeorado según los parámetros ingresados al modelo. Esto último deja muestra de la integra dependencia de la calidad de los resultados con la configuración de la herramienta.

Finalmente, entre los trabajos futuros planteados, se encuentran:

- Determinación de expresiones analíticas para la función peso dependientes del idioma y el objeto de búsqueda.
- Adaptación del modelo presentado para textos de gran volumen.
- Determinación de métodos de aprendizaje para la eliminación de nodos del grafo.

Referencias

[1] Atkinson-Abutridy, J.; Mellish, C.; Aitken,May-Jun 2004, S. Combining information extraction with genetic algorithms for text mining, Intelligent Systems Volume 19, Issue 3.

[2] Cristobal Baray y Kyle Wagner, 1999, "Where Do Intelligent Agents Come From?". Crossroad ACM.

[3] Claude Berge, 1962, "Teoria de las redes y sus aplicaciones". Cia. Ed. Continental, S. A. México-España.

[4] V. Julián, V. Botti. 2000. "Agentes Inteligentes: el siguiente paso en la Inteligencia Artificial". NOVATICA / may.-jun. 2000 / Especial 25 aniversariom Edici.n digital / © ATI 2000 95

[5] J.A. Boundy, U.S.R. Murty, 1976, "Graph Theory with applications", North Holland, ISBN: 0-444-19451-7

[6] Bratman, M., 1987, "Intentions, Plans, and Practical Reason". Harvard Univ. Press. ISBN: *1575861925*

[7] Enrique Chacon, 1973, "Teoría de los grafos, Investigación operativa", La empresa moderna, ISBN: 84-256-0204-1

[8] Chen, 2002,"Discrete mathematics",Electronic Edition, www.maths.mq.edu.au/~wchen/lndmfolder/lndm.html

[9] Junghoo Cho, Sridhar Rajagopalan, 2001, A Fast Regular Expression Indexing Engine, http://citeseer.ist.psu.edu/cho01fast.html (dic-2007)

[10] Ciravegna et al., Ed.,2001, Proc. of the 17Th International Joint Conference on Artificial Intelligence (IJCAI-2001), Workshop of Adaptive Text Mining, Seattle, WA.

[11] *Federico F., Ale, J. M., "Aplicación de la teoria de agentes al modelo de grafos para detección de patrones en textos", Octubre 2007, CACIC 2007, Workshop de Agentes y Sistemas Inteligentes, 1347-1358*

[12] Feldman, Ed., 1999, Proc. of The 16th International Joint Conference on Artificial Intelligence (IJCAI-1999), Workshop on Text Mining: Foundations, Techniques and Applications, Stockholm, Sweden

[13] Iglesias Fernández, C. A. (1998). Definición de una metodología para el desarrollo de sistemas multiagente. PhD thesis, Departamento de Ingeniería de Sistemas Telemáticos. Universidad Politécnica de Madrid.

[14] Stan Franklin y Art Graesser, 1996, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag.

[15] Carrero García, F., Gómez Hidalgo, J.M., Puertas Sanz, E., Maña López, M., Mata, J. , 2007, Attribute Analysis in Biomedical Text Classification. Second BioCreAtIvE Challenge Workshop: Critical Assessment of Information Extraction in Molecular Biology, Spanish Nacional Cancer Research Centre (CNIO), Madrid, SPAIN, April 23rd-25th

[16] Michael Georgeff,Barney Pell, Martha Pollack,Milind Tambe, MichaelWooldridge, 1999, "The Belief-Desire-Intention Model of Agency", Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)

[17] Ralph P. Grimaldi,1998,"Matematicas discreta y combinatoria, Una introducción con aplicaciones" 3era Ed., Addison Wesley, ISBN:968-444-324-2

[18] Wei Jin, Rohini Srihari, 2007, "Graph-based text representation and knowledge discovery", Symposium on Applied Computing archive Proceedings of the 2007 ACM symposium on Applied computing table of contents, 807 - 811 , ISBN: 1-59593-480-4

[19] Junji T., Hidekazu N., Megumi I., ,Mayo 2004, "Graph-based text database for knowledge discovery" Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters WWW Alt. '04, ACM Press

[20] David Kinny, Michael George, Anand Rao, 1996, "A Methodology and Modelling Technique for Systems of BDI Agents ", Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world, , 56-71, ISBN:3-540-60852-4 Environmental Research Institute of Michigan (ERIM)

[21] David Kinny,Michael George, Anand Rao, 1996,"A Methodology and Modelling Technique for Systems of BDI Agents", Technical Report 59, Australian Artificial Intelligence Institute, Melbourne, Australia.

[22] Tsuyoshi Kitani, Yoshio Eriguchi, Masami Hara, 1994, "Pattern Matching and Discourse Processing in Information Extraction from Japanese Text ", Journal of Artificial Intelligence Research 2, ,89-110

[23] Chung-Hong Lee; Hsin-Chang Yang,. A classifier-based text mining approach for evaluating semantic relatedness using support vector machines, ITCC 2005. International Conference on Information Technology: Coding and Computing, 2005 Volume 1, Issue , 4-6 April 2005 Page(s): 128 - 133 Vol. 1

[24] Giuseppe. Liotta, September 21-24,2003, "Graph Drawing" 11th International Symposium, Gd 2003, Perugia, Italy - Mathematics – 2004

[25] Christopher D. Manning, Hinrich Schütze, 1999, Foundations of Statistical Natural Language Processing, MIT Press: 1999. ISBN 0262133601.

[26] Christopher D. Manning, Hinrich Schütze, 1999, Foundations of Statistical Natural Language Processing, MIT Press: 1999. ISBN 0262133601.

[27] M. Marko, M. A. Porter, A. Probst, C. Gershenson, A. Das, 2002, "Transforming the World Wide Web into a Complexity-Based Semantic Network", http://arxiv.org/html/cs/0205080

[28] Mladenic, Ed., 2000, Proc. of the Sixth International Conference on Knowledge Discovery and Data Mining, Workshop on Text Mining, Boston, MA.

[29] Mladenic, Ed. (2000), Proc. of the Sixth International Conference on Knowledge Discovery and Data Mining, Workshop on Text Mining, Boston, MA, 2000.

[30] H. Van Dyke PARUNAK. 1998. "Practical and Industrial Applications of Agent-Based Systems",

[31] Martin Rajman, Romaric BESANÇON, (1997) Oct 7-10, Proceedings of the seventh IFIP 2.6 Working Conference on Database Semantics (DS-7), Chapam & Hall IFIP Proceedings serie

[32] Russell, S. Norving, P. 1995. "Artificial Intelligence: A Modern Approach". Prentice-Hall.,912 pp, SBN-13: 9780131038059

[33] Jorge J. Gómez Sanz, 2003, "Metodologías para el desarrollo de sistemas multiagente", Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial, ISSN 1137-3601, Nº. 18, 2003, pags. 51-64

[34] Swapan Kumar Sarkar, 2005, "A Textbook of Discrete Mathematics", Chand (S.) & Co Ltd ,India, 768 pages, ISBN: 978-8121922326

[35] Fabrizio Sebastiani, (2002), Machine Learning in Automated Text Categorization, ACM Computing Surveys

[36] Mark Sharp, 11 December 2001, Text Mining, Seminar in Information Studies, Prof. Tefko Saracevic.

[37] Jan Sudeikat, Lars Braubach, Alexander Pokahr, Winfried Lamersdorf and Wolfgang Renz, 2007 , "On the Validation of Belief–Desire–Intention Agents.", Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 185-200,ISB: 978-3-540-71955-7

[38] W. H. Tsai y K. S. Fu. , Dec. 1979, "Error-correcting isomorphisms of attributed relational graphs for pattern analysis," IEEE Trans. Syst., Man, Cyhern.. vol. SMC-9, no. I ? , pp. 757-768.

[39] WH Tsai, KS Fu, Jan. 1983, "Subgraph error correcting isomorphisms for syntactic pattern recognition," IEEE Trans. Sysr., Man, Cybern., vol. SMC-13, no. I , pp. 48-62.

[40] Springer-Verlag Heidelberg, 2005, "Graph Theory", Electronic edition. http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.counted.pdf

[41] Romain Vinot, Francois Yvon, 2003, Improving Rocchio with Weakly Supervised Clustering, proceedings of the 14th European Conference on Machine Learning, ECML 2003, Pag: 456.

[42] Pedro Isasi Viñuela, Paloma Martinez Fernández, Daniel Borrajo Millán, 1997, "Lenguajes, Grámaticas y Autómatas, un enfoque práctico", Addison-Wesley, ISBN:84-7829-014-1

[43] A. K. C. Wong y S . W. Lu and M. Rioux, Sept. 1989, "Recognition and shape synthesis of 3-D objects based on attributed hypergraphs," IEEE Trum. Purrern Anal. Machine Intell., vol. PAMI-1 1, no. 3, pp. 279- 289.

[44] Pak Chung Wong, Paul Whitney, Jim Thomas, 1999, Visualizing Association Rules for Text Mining, Proceedings of the 1999 IEEE Symposium on Information Visualization Page: 120, ISBN:0-7695-0431-0

[45] Dong-Qing Zhang, Shih-Fu Chang., 2004, "Stochastic Attributed Relational Graph Matching for Image Near-Duplicate Detection", DVMM Technical Report #206-2004-6, Department of Electrical Engineering, Columbia University

APENDICE

Especificación de clases: Paquete UI - UIProyectManager

Clase que se encarga de manejar la interfaz principal del proyecto. Posee métodos para manejo de los archivos del proyecto e inicia el proceso de entrenamiento y evaluación.

Atributos:

Parameters* __param;

Puntero a la clase Parameters

Graficos* __graf;

Puntero a la clase UIGraph

String* __folder;

Carpeta de trabajo

String* __projName;

Nombre del proyecto

Thread* __trd;

Thread donde se ejecuta el entrenamiento y la Evaluación

Métodos:

void StartProyect();

Inicializa el proyecto.
Tanto en el aspecto grafico (manejo de los controles) como en lo relacionado con los archivos necesarios para entrenamiento y evaluación.

void SaveProyect_f();

Guarda el estado actual del proyecto.

void setFolder(System::String* _f);

Setea la carpeta de trabajo a las clases para el manejo de parámetros y gráficos.

Parámetros:
String* _f: Carpeta de trabajo.

void Training();

Ejecuta la acción de entrenamiento. Llama a los paquetes de procesamiento para analizar los textos de entrenamiento.

void Evaluate();

Evalúa los textos dispuestos en la tabla de tareas. Llama a los métodos de procesamiento de textos.

void SetEnable(bool _b);

Determina el estado de los controles del formulario.

Parámetros:
bool _b: valor booleano que determina si el proyecto
esta activo o no.

void AddToLog(System::String* _s);

Agrega al Log es String que se pasa por parámetro.

Parámetros:
String* _s: Mensaje para agregar al Log.

unsigned int ReadFile(String* __file, ParserBuilder* pb, config* _conf,String* __type);

Ejecuta los métodos de las clases de procesamiento de texto para leer un archivo que se encuentre en la lista de actividades.
Devuelve la cantidad de palabras procesadas.

Parámetros:
String* __file: Nombre del archivo.
ParserBuilder* pb: Factory de parser.
String* __type: Tipo de archivo (clasificación).

int setParameters(config* _c, String* _id);

Carga en la estructura _c los parámetros identificados con el _id.
Devuelve 0 si el proceso fue ejecutado con normalidad.

Parámetros:
config* _c: Estructura que almacena los parámetros definidos por el usuario.
String* _id: Id de los parámetros del usuario.

Especificación de clases: Paquete UI - Parameters

Clase que se encarga de manejar la interfaz asociada a la definición de los parámetros del modelo por parte del usuario.

Atributos:

String* __folder;

Carpeta de trabajo

int _index;

Índice de parámetro

ZedGraph::ZedGraphControl * zedGraphControl1;

Gráfico

int _action;

Acción realizada (borrar, guardar, nuevo)

Métodos:

void fillTxtBox();

Llena los cuadros de texto del formulario con la información de los parámetros almacenada en disco.

void SetEnable(bool v1);

Determina el estado de los controles en función de que si el formulario debe estar habilitado o no.

Parámetros:
bool v1: Determina si el formulario está habilitado o no.

void CleanTextBox();

Limpia todos los cuadros de texto donde se muestra información.

int Save(int _idx);

Guarda los parámetros cargados por el usuario.

void Initialize();

Inicializa el formulario.

void SaveData();

Guarda el último estado de los parámetros en disco.

void Close();

Cierra el formulario.

void LoadData();

Lee la información almacenada en disco.

void SetFolder(System::String* _f);

Carga la carpeta de trabajo.

System::Data::DataTable* GetData();

Devuelve la información vinculada con los parámetros del modelo.

Especificación de clases: Paquete UI - UIGraph

Clase que se encarga de manejar la interfaz asociada a los gráficos de evaluación de tiempo del modelo.

Atributos:

String* __folder;

Carpeta de trabajo.

ZedGraph::ZedGraphControl * graph;

Grafico.

Métodos:

void addTimeValue(unsigned int __wc,double __time);

Agrega un tiempo de evaluación del modelo a la tabla que contiene esta información y que luego se grafica.

Parámetros:
unsigned int __wc: Cantidad de palabras procesadas
double __time: Tiempo empleado.

void SetFolder(String* _f);

Setea la carpeta de trabajo.

Parámetros:
String* _f: Carpeta de trabajo.

void LoadData();

Lee la información del proyecto almacenada en disco.

void SaveData();

Guarda la información de los tiempos en disco.

void Close();

Cierra el formulario de trabajo

Especificación de clases: Paquete Math - Graph

Clase que se encarga de manejar la configuración de los gráficos utilizados en el proyecto.
Clase abstracta. Define interfaz.

Atributos:

Métodos:

virtual void setGraphParameters(ZedGraphControl* graph, DataTable* table);

Setea los parámetros necesarios para poder trabajar con el control ZedGraphControl.

Parámetros:
ZedGrapfControl* graph: Grafico al que se le desea establecer los parámetros.
DataTable* table: Conjunto de datos que se deben registrar en el control.

Especificación de clases: Paquete Math - TimeGraph

Clase que se encarga de manejar la configuración de los gráficos utilizados en el proyecto.

Implementación de la interfaz definida por la clase Graph para los gráficos de tiempo.

Atributos:

Métodos:

void setGraphParameters(ZedGraphControl* graph, DataTable* table);

Método definido en Graph. Implementación para gráficos de tiempo.

Especificación de clases: Paquete Math - FunctionGraph

Clase que se encarga de manejar la configuración de los gráficos utilizados en el proyecto.

Implementación de la interfaz definida por la clase Graph para los gráficos de función de evaluación.

Atributos:

Métodos:

void setGraphParameters(ZedGraphControl* graph, DataTable* table);

Método definido en Graph. Implementación para gráficos de función de evaluación.

Especificación de clases: Paquete Math - GraphBuilder

Builder de las clases Graph. Implementa patrón builder.

Atributos:

TimeGraph* __tg;

Puntero a la clase TimeGraph

FunctionGraph* __fg;

Puntero a la clase FuncionGraph

Métodos:

Graph* getGraph(String* _tipo);

Devuelve un puntero a Graph.

Parámetros:
String* _tipo: Tipo de grafico.

graphBuilder();

Constructor

Especificación de clases: Paquete Math - Function

Parser y evaluador de funciones.

Atributos:

ExpressionParser* __fParser;

Puntero a clase que parsea funciones (no partidas)

Collections::ArrayList* vFunctions;

Colección de funciones (en caso de función partida)

unsigned int _error;

Código de error.

HashTable* h;

Variables

Métodos:

int FindItem(double _x);

Busca la función a evaluar en la colección según el valor que se pasa por parámetro.

Parámetro:
double _x: valor donde se quiere evaluar la función.

void SetParser(int i,double _x);

Carga la configuración de la clase MathParser

Parámetros:
int i: Índice de función.
double _x: valor a evaluar.

Function();

Constructor.

void SetFunction(String* _func);

Carga la función que se desea evaluar (cualquier función incluso partidas)

Parámetros:
String* _func: Expresión de la función.

double GetValueAsDouble(double _x);

Devuelve la imagen del valor que se pasa por parámetro (double)

Parámetros:
double _x: Valor en x.

String* GetValueAsSring(double _x);

Devuelve la imagen del valor que se pasa por parámetro (String)

Parámetros:
double _x: Valor en x.

unsigned int GetError();

Devuelve el código de error.

Especificación de clases: Paquete PDM - PDM

Clase que define la interfaz del modulo de detección de patrones. Funciona como facade de todo el modulo. Clase Abstracta.

Atributos:

MGRAMS _gv;

Colección de gramas del texto

uint _ti;

Contador de iteraciones (util para calcular serie)

GCACHE _vc;

Cache de gramas (contiene los últimos gramas procesados)

Métodos:

pdm(uint _cs);

Constructor

Parámetros:
unsigned int _cs: tamaño de la cache.

virtual void notify_gram(std::string& _s, uint _pos);

Notifica la aparición de un grama en el texto.

Parámetros:
Sring _s: Grama.
unsigned int _pos: Posición del grama en el texto.

virtual void notify_new_txt();

Notifica que se va a procesar un nuevo texto.

virtual void get_patterns(unsigned int _noW,std::string __file);

Devuelve los patrones detectados como un listado en el archivo que se pasa por parámetro.

Parámetros:
unsigned int _noW: Numero de palabras procesadas.
string __file: Nombre de archivo que se desea escribir.

Especificación de clases: Paquete PDM - Cache

Concreto de PDM

Atributos:

GramBuilder* _gb;

Builder de gramas

static cpdm* _instance;

Instancia del CPDM (singleton)

Métodos:

void add_pattern(CN _g);

Agrega un nuevo grama a la cache.

Parámetros:
CN _g: Grama.

cpdm(uint _cs);

Constructor

static cpdm* instance(uint _cs);

Devuelve la instancia de CPDM

void notify_gram(std::string& _s, uint _pos);
void notify_new_txt();
void get_patterns(unsigned int _noW,std::string __file);

Métodos definidos en la clase PDM

Especificación de clases: Paquete PDM - Cache

Template de cache de tamaño fijo.

Atributos:

uint sz;

Tamaño de la cache.

uint idx;

Índice en la cache.

uint ne;

Número de elementos insertados en la cache.

Métodos:

cache (uint s);

Constructor.

Parámetros:
unsigned int s: Tamaño de la cache.

cache ();

Constructor default.

void set_size(uint size);

Define el tamaño de la cache.

Parámetros:
unsigned int size: Tamaño de la cache.

T& operator [](uint element)const;

Obtiene el iesimo elemento de la cache.

Parámetros:
unsigned int element: Índice del elemento que se desea.

void insert(const T &t);

Agrega un elemento a la cache.

Parámetros:
const T &t: Elemento a agregar.

inline uint get_size()const;

Obtiene el tamaño de la cache.

inline void clear();

Limpia toda la cache.

Especificación de clases: Paquete PDM - ArcBuilder

Factory de arcos del grafo.

Atributos:

static ArcBuilder* _instance;

Instancia de l ArcBuilder (Singleton)

Métodos:

ArcBuilder();

Constructor

static ArcBuilder* instance();

Obtiene la instancia del ArhBuilder

arcs* BuildArc(vertex* _v,uint _p1,uint _p2);

Construye un nuevo arco.

Parámetros:
vertex* _v: vértice del grafo.
unsigned int _p1: Posición del vértice que conecta.
unsigned int _p2: Posición del vértice que conecta.

void DestroyArc(arcs* a);

Elimina el arco que se pasa por parámetro.

Parámetros:
arcs* a: Arco a eliminar.

Especificación de clases: Paquete PDM - GramBuilder

Factory de vértices del grafo.

Atributos:

static GramBuilder* _instance;

Instance del GramBuilder (Singleton)

ArcBuilder* _ab;

Factory de arcos

Métodos:

GramBuilder();

Constructor

static GramBuilder* instance();

Obtiene la instancia del GramBuilder

vertex* BuildGram(std::string _s, uint _pos);

Constructor de Grama.

Parámetros:
String _s: grama.
unsigned int _pos: Posición en el texto.

Especificación de clases: Paquete PDM - PDMFactory

Factory para crear al pdm.

Atributos:

static ArcBuilder* _arcBuild;

Factory de Arcos.

static GramBuilder* _gramBuild;

Factory de vertices.

static cpdm* _cpdm;

Instancia del cpdm (Singleton).

Métodos:

static pdm* CreatePDM(config* conf);

Crea el PDM. Determina todo el ambiente de trabajo a partir de la estructura de configuración que se pasa por parámetro.

Parámetros:
config* conf: Estructura de configuración del PDM

static void DestroyPDM();

Destruye todo el ambiente de trabajo.

Especificación de clases: Paquete PDM - Gram

Clase abstracta para manejo de gramas.

Atributos:

string _gram;

Grama

Métodos:

gram(std::string& g);

Constructor

Parámetros:
String& g: grama.

virtual uint notify_position(uint lp,GCACHE& c);

Registra cambio de posición del grama en el texto. Actualiza los pesos del arco.

Parámetros:
unsigned int lp: Ultima posición.
GCACHE& c: Cache para actualizar pesos.

inline virtual std::string get_ID()const;

Obtiene el ID del grama

Especificación de clases: Paquete PDM - Vertex

Clase para manejo de vértices del grafo.

Atributos:

SS _ss;

Estructura para calcular la serie para eliminación de arcos.

uint _pos;

Última posición del grama

double _f;

Frecuencia

uint _sc;

Contador de iteraciones

GL _glist;

Lista de arcos asociados al vértice

static ArcBuilder* _ab;

Constructor de arcos

Métodos:

void modify_weights(uint lp,GCACHE& c);

Modifica los pesos de los arcos.

Parámetros:
unsigned int lp: Última posición.
GCACHE& c: Cache de gramas

inline bool serie_ok();

Consulta a la serie si debe realizar una eliminación de arcos.

vertex(std::string& g,uint pos,double _frec=1);

Constructor.

Parámetros:
string& g: Grama.
unsigned int pos: Posición.
double _frec: frecuencia.

static void set_ArcBuilder(ArcBuilder* ab);

Setea la factory de arcos.

Parámetros:
ArcBuilder* ab: Puntero a ArcBuilder.

uint notify_position(uint lp,GCACHE& c);

Definido en Grams.

inline virtual uint get_position()const ;

Obtiene la ultima posición del grama.

void getRelations(std::set<vertex*>& vs);

Obtiene el conjunto de vértices con los que se relaciona

Especificación de clases: Paquete PDM - Arcs

Clase para manejo de vértices del grafo.

Atributos:

double _w;

Peso del arco

uint _f;

Frecuencia del arco

char _state;

Estado

vertex *_v;

vértice adyacente

Métodos:

WI refresh_weight(uint pos1,uint pos2);

Modifica el peso del arco.

Parámetros:
unsigned int pos1: Posición vértice adyacente.
unsigned int pos1: Posición vértice adyacente.

void initilialize(vertex *v, uint pos1,uint pos2);

Inicializa el arco cuando se crea.

Parámetros:
vertex *v: vértice adyacente.
unsigned int pos1: Posición vértice adyacente.
unsigned int pos1: Posición vértice adyacente.

inline void force_weight(double w);

Fuerza el valor del peso del arco (setter)

Parámetros:
double w: peso del arco.

arcs(vertex* _ve,uint _p1,uint _p2);

Constructor.

Parámetros:
vertex *v: vértice adyacente.
unsigned int pos1: Posición vértice adyacente.
unsigned int pos1: Posición vértice adyacente.

Especificación de clases: Paquete ProxyDB - dbManager

Clase que controla el acceso a la BD.

Atributos:

MySqlConnection* conn;

Conexión con la base de datos.

String* ConnectionString;

String de conexión.

Métodos:

void connect();

Conexión con MySQLServer

void close();

Cierre de Conexión

void createDB(String* DBName);

Crea la base de datos

Parámetros:
String* DBName: Nombre de la BD.

void createTable(String* __type);

Creación de tablas.

Parámetros:
String* __type: Nombre de tabla.

System::Collections::ArrayList* getTables();

Obtiene todas las tablas existentes en la BD.

void dropTables(System::Collections::ArrayList* Tables);

Elimina todas las tablas que se pasan por parámetro.

Parámetros:
ArrayList* Tables: Listado de Tablas a borrar.

dbManager();

Constructor. Solo conecta con el Servidor de BD.

dbManager(String* DBname);

Constructor. Se conecta al servidor y crea la BD.

Parámetros:
String* DBName: Nombre de la Base de datos.

dbManager(Data::DataTable* ActivityTable, String* DBname);

Constructor. Se conecta al servidor. Crea la base de datos y sus tablas.

Parámetros:
DataTable* ActivityTable: Lista de actividades a realizar por el sistema.
String* DBName: Nombre de la base de datos.

bool checkDatabase(String* DBName);

Chequea si existe la base de datos

Parámetros:
String* DBName: Nombre de la base de datos.

void fillData(String* __table);

Llena la tabla que se pasa por parámetro con la información de un archivo llamado __table.txt

Parámetros:
String* __table:Nombre de la BD

void difTables(Collections::ArrayList* Tables);

Obtiene la diferencia de cada una de las tablas de la base de datos. Almacena cada resultado en una tabla nueva.

Parámetros:
ArrayList* Tables: Nombres de Tablas.

void prepareDB();

Crea la base de conocimiento del Sistema.

void eraseDB();

Borra todas las tablas de la BD.

void fillEvalData(String* type);

Carga los datos despues de la evaluación de un texto.

Parámetros:
String* type: Tipo de texto.

System::Collections::ArrayList* getEvalIntersection();

Obtiene la intersección entre la tabla de Evaluación y el resto. Devuelve un resumen con la cantidad de registros en común con la tabla de evaluación de cada tabla de conocimiento.

Especificación de clases: Paquete NLP - Parser

Clase abstracta que define la interfaz de un parser.

Atributos:

NlpBuilder* nlpBuild;

Builder de clase de Procesamiento Natural

String* __words[];

Conjunto de palabras del texto

int __WIdx;

Índice de la ultima palabra accedida

String *file;

Ruta del archivo a procesar

Métodos:

virtual void LoadFile(String *_f);

Lee el archivo desde disco

Parámetros:
String* _f: archivo

virtual void GetWordCollection(String* __txt);

Carga la colección de palabras a partir de un texto que se pasa por parámetro. El texto debe estar limpio de todo Tag.

Parámetros:
String* __txt: texto

Parser(System::String *_file);

Constructor.

Parámetros:
String* _file: Nombre de archivo.

virtual String* get_word();

Devuelve una palabra del texto (según _Widx)

virtual void reset();

Limpia al parser de toda información.

Especificación de clases: Paquete NLP - HTMLParser

Parser HTML.

Atributos:

Cllections::ArrayList* __AH;

Colección de caracteres ASCII y su correspondencia a ISO

String *__HTML;

Texto completo con TAGs HTML

Métodos:

void loadHTML();

Lee el HTML de disco

void ReadAsciiConf();

Le el archivo de configuración de caracteres ASCII (archivo interno del Sistema)

String* ISOtoAscii(String* __txt);

Cambia los caracteres ASCII a ISO

Parámetros:
String* __txt: Texto del cual se quiere eliminar los caracteres ASCII.

String* GetExpressionForTagContents(String* __tag);

Obtiene la información dentro de un TAG.

Parámetros:
String* __tag: TAG
String* get_body();

String* get_txt();

Limpia el documento de TAGs.

String* StripComments(String* _txt);

Elimina los comentarios.

Parámetros:
String* _txt: Texto de donde hay que eliminar comentarios.

String* get_word();

Definido en Parser.

void LoadFile(String* _f);

Definido en Parser.

HTMLParser(String *_file);
HTMLParser();

Constructor

Especificación de clases: Paquete NLP - HTMLParser

Parser HTML.

Atributos:

Collections::ArrayList* __AH;

Colección de caracteres ASCII y su correspondencia a ISO

String *__HTML;

Texto completo con TAGs HTML

Métodos:

void loadHTML();

Lee el HTML de disco

void ReadAsciiConf();

Le el archivo de configuración de caracteres ASCII (archivo interno del Sistema)

String* ISOtoAscii(String* __txt);

Cambia los caracteres ASCII a ISO

Parámetros:
String* __txt: Texto del cual se quiere eliminar los caracteres ASCII.

String* GetExpressionForTagContents(String* __tag);

Obtiene la información dentro de un TAG.

Parámetros:
String* __tag: TAG

String* get_body();

Obtiene el cuerpo del Documento HTML

String* get_txt();

Limpia el documento de TAGs.

String* StripComments(String* _txt);

Elimina los comentarios.

Parámetros:
String* _txt: Texto de donde hay que eliminar comentarios.

String* get_word();

Definido en Parser.

void LoadFile(String* _f);

Definido en Parser.

HTMLParser(String *_file);
HTMLParser();

Constructor

Especificación de clases: Paquete NLP - ParserBuilder

Factory de la Clase Parser (Patrón Builder)

Atributos:

HTMLParser* __htmlP;

Puntero a HTMLParser

TxtParser* __txtP;

Puntero a TxtParser

unsigned int __state;

Código de error

Métodos:

Parser* getParser(String* __file);

Devuelve un parser en función del archivo que se quiere procesar.

Parámetros:
String* __file: Nombre de archivo a procesar.

inline unsigned int getState();

Devuelve código de error.

ParserBuilder();

Constructor

Especificación de clases: Paquete NLP - EnglishDictionary

Clase que interpreta palabras del idioma ingles

Atributos:

Métodos:

int getWordType(String* Type);

Detecta el tipo de palabra de acuerdo a los tipos que se
quieran procesar. El método devuelve el tipo de palabra sin conjugación.

Ejemplo: Verbo en pasado -> Tipo de palabra: Verbo.

Parámetros:
String* type: TAG que define el tipo de palabra y su conjugación

String* ReadDictionary(String* Criteria);

Método que lee el diccionario con un criterio en SQL.
Si lo buscado no se encuentra en el diccionario, entonces devuelve nulo.

Parámetros:
String* Criteria: Criterio SQL

String* convertVerb(String* verb,String* type);

Cambia la conjugación de los verbos regulares y en presente simple

Parámetros:
String* verb: Verbo a convertir
String* type: Conjugación

String* verbAnalisis(String* relation[]);

Estudia los tiempos verbales y los modifica llevándolo a su forma base

Parámetros:
String* __gc[]: Relación entre los verbos y su forma base (según conjugación)

String* nounAnalisis(String* relation[]);

Estudia los sustantivos y los devuelve en su forma base

Parámetros:
String* __gc[]: Relación entre los sustantivos y su forma base (según conjugación)

String* convertNoun(String* noun,String* type);

Devuelve un Sustantivo en su forma base

Parámetros:
String* noun: Sustantivo a convertir
String* type: Conjugación

String* advAnalisis(String* relation[]);

Analiza los adverbios y los devuelve en su forma base

Parámetros:
String* __gc[]: Relación entre los adverbios y su forma base (según conjugación)

String* adjAnalisis(String* relation[]);

Analiza los Adjetivos y los devuelve en su forma base

Parámetros:
String* __gc[]: Relación entre los adjetivos y su forma base (según conjugación)

EnglishDictionary();

Constructor.
Crea la conexión con el diccionario.

String* getWordRoot(String* word);

Método que a partir de una palabra y su tipo devuelve la misma palabra en
su formato base. (en caso de no detectar la forma base se devuelve la misma palabra)

Parámetros:
String* word:Palabra de la que se quiere su forma base.

Especificación de clases: Paquete NLP - NLP

Clase Abstracta. Define la interfaz de un NLP

Atributos:

Métodos:

virtual String* getTextRoot(String* text) __gc[];

Método que devuelve la base de palabras del texto

Parametros:
String* text: Texto a procesar.

virtual String* getTextWithTags(String* text);

Método que agrega al texto los identificadores de cada palabra

Parámetros:
String* text: Texto a procesar.

Especificación de clases: Paquete NLP - EnglishNLP

Clase que permite procesar los textos en ingles.

Atributos:

MaximumEntropySentenceDetector* mSentenceDetector;
EnglishMaximumEntropyTokenizer* mTokenizer;
EnglishMaximumEntropyPosTagger* mPosTagger;

Atributos de la liberaría OpenNLP

String* mModelPath;

Dirección del modelo (diccionario para interpretar idioma)

Métodos:

String* SplitSentences(String* text) __gc[];

Método que selecciona solo las oraciones.

Parámetros:
String* text: Texto a procesar.

String* GetTokens(String* text) __gc[];

Separador de palabras y símbolos

Parámetros:
String* text: Texto a procesar.

String* GetTag(String* Tokens __gc[]) __gc[];

Asigna los tags a cada palabra

Parámetros:
String* Tokens[]: Conjunto de palabras.

EnglishNLP();

Constructor. Se le pasa por parámetro el path donde se encuentran los archivos de diccionario para interpretar las palabras.

String* getTextWithTags(String* text);

Devuelve el texto con los Tags correspondientes

Parámetros:
String* text: Texto a procesar.

String* getTextRoot(String* text) __gc[];

Devuelve una colección de palabras en su forma base

Parámetros:
String* text: Texto a procesar.

Especificación de clases: Paquete NLP - NLPBuilder

Factory de NLP

Atributos:

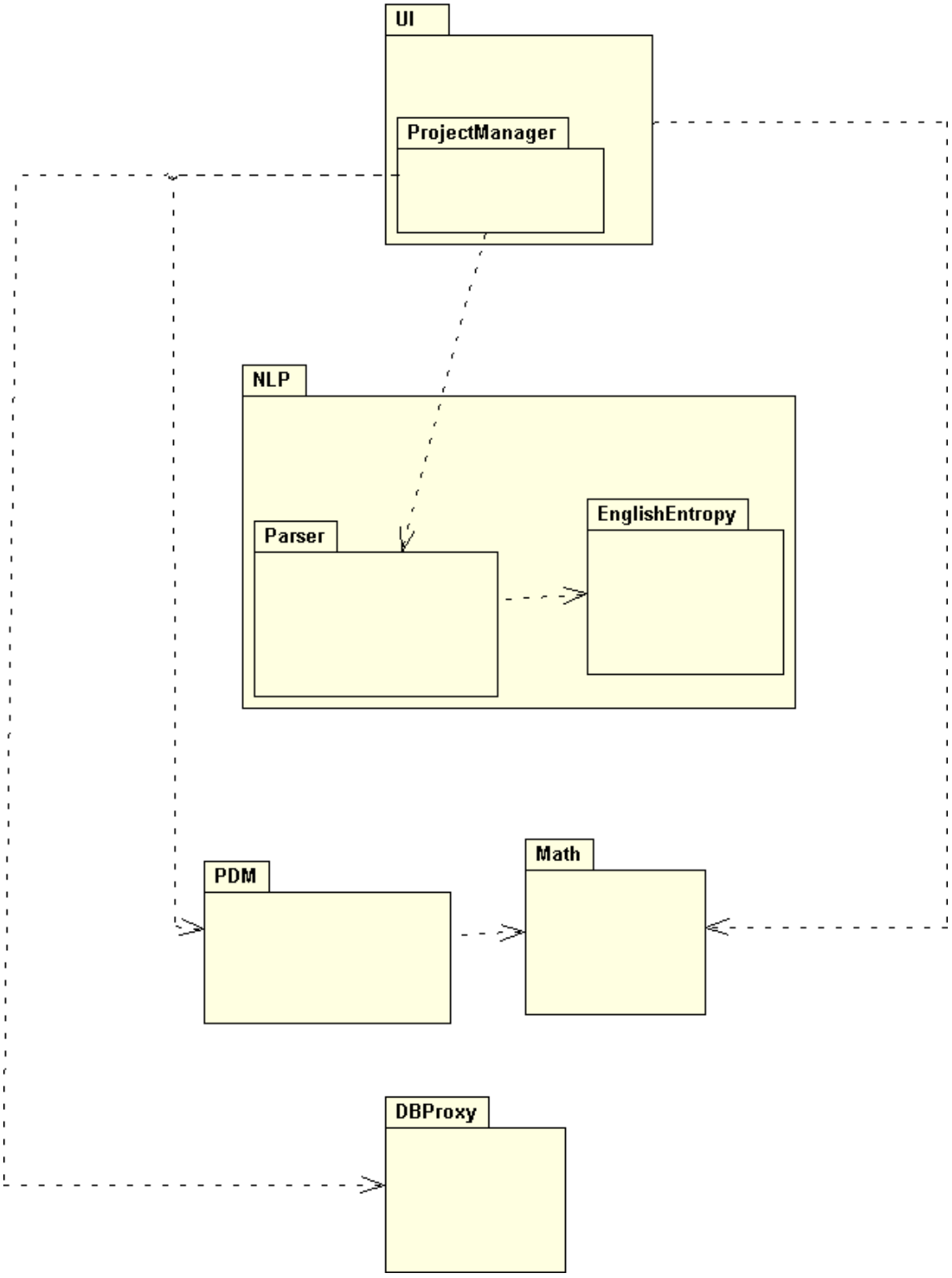
Métodos:

NLP* getNLP(String* lang);

Obtiene una instancia del procesador de lenguaje natural.

Parámetros:
String* lang: Lenguaje del archivo a procesar

Paquetes


UI
ProjectManager
NLP
Parser
EnglishEntropy
PDM
Math
DBProxy

Componentes