FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA

DEPARTAMENTO DE COMPUTACIÓN

Sistema de Generación de Texto Automático en Dominios Acotados

Tesis realizada por Bridera Claudio Daniel para la Licenciatura en Ciencias de la Computación en la Universidad Nacional de Córdoba

---

Dirigida por:
Doctor Martín Ariel Domínguez
Doctor Pablo Ariel Duboue

Con la colaboracion de:
Licenciado Demetrio Martín Vilela

Agradecimientos

- A mis padres Ramón y Anita, quienes siempre serán los primeros agradecidos en cualquier logro de mi vida por formarme como persona, por el amor y el apoyo incondicional que siempre me brindan.
- A mis hermanos Gustavo y Laura, a mi cuñado Diego, a mi novia Melina y a mis sobrinos Eileen Y Emmanuel por todo el apoyo y cariño que me dan día a día.
- A mis directores Martín y Pablo por la calidez humana y la gran ayuda que siempre me prestaron a lo largo del proyecto.
- A Demetrio Vilela por colaborar activamente con el trabajo, por ser una gran persona, profesor y amigo.
- A mis amigos y compañeros Kevin, Fer, Franco, Negro, Manu, Lara, Maxi, Fede, Gonza, Alan y Agus por tantas vivencias, asados y metegoles compartidos a lo largo de estos maravillosos cinco años.
- Al Ingeniero Marcelo Cometto y a todo el equipo APDB (Eric, Manu) por ser tan grandiosas personas, por darme la posibilidad de crecer profesionalmente y confiar en mi.
- A los señores Oscar Vergara y Oscar Maldonado con quienes tuve el placer de trabajar, por sus enseñanzas de la vida que siempre llevaré presente.

Resumen

El presente trabajo consiste en desarrollar un sistema de generación de lenguaje natural basado en templates capaz de producir avisos clasificados bien formados referidos a un producto en proceso de comercialización o de promoción. Para tal objetivo elaboramos un corpus, extrayendo de Internet avisos clasificados, el que estudiamos y analizamos utilizando diversas técnicas de procesamiento de texto. A lo largo de este trabajo describimos el desarrollo de dos algoritmos particulares implementados en Php: AdGen y su componente principal AdTagger. El primero es un sistema capaz de aprender estadísticamente representaciones de plantillas para la generación de texto, y el segundo es un etiquetador de entidades de dominio que posee una precisión del 89 % y fue elaborado en base al corpus disponible.
Identificamos cuáles entidades del dominio son determinantes para la construcción de avisos clasificados y además mostramos cómo los errores de redacción del cospus como así también la estructura gramatical de los avisos afectan el proceso de generación.

Palabras Claves: Generación de Lenguaje Natural, P.O.S Tagging, Clustering

Clasificación: I.2.7 Natural Language Processing. Language generation

Índice general

Introducción

La generación de lenguaje natural es una tarea sumamente compleja y ha sido investigada en numerosas publicaciones (como [5],[6],[7] y [11])para su aplicación en diversas áreas. Puede definirse como el proceso de construir o elaborar un texto en lenguaje natural para la comunicación con fines específicos.

Por lo general, un generador de texto tiene acceso a un gran conjunto de datos del cual ha de seleccionar información para presentar a los usuarios un texto legible y semánticamente fiel. Generar texto es, pues, un problema de toma de decisiones con múltiples restricciones: de conocimiento proposicional, de herramientas lingüísticas disponibles y de las intenciones de quien produce el texto. Se trata de identificar los factores involucrados en este proceso y de determinar la mejor forma de representar estos factores y sus dependencias.

Una de las áreas en las que la generación del lenguaje natural ha encontrado numerosas aplicaciones en los últimos años es el marketing y en particular etapas relacionadas con la promoción de productos que están en pleno proceso de comercialización. Con el auge de las redes sociales y los sitios de compraventa, se ha desatado una carrera entre las empresas dedicadas a publicitar vía Internet sus productos para acaparar usuarios y clientes. Éstas recurren a diversas técnicas de generación de texto con el fin de poblar de manera coherente sus correspondientes sitios. Entre las tareas que podemos mencionar se encuentra la descripción de un producto de manera automática respetando las limitaciones y restricciones gramaticales que presenta un idioma. Esta tarea supone importantes esfuerzos algorítmicos para lograr un contenido estéticamente bien logrado.

El presente proyecto consiste en desarrollar un sistema capaz de generar textos bien formados referidos a un producto en proceso de comercialización o de promoción. El sistema será capaz de aprender estadísticamente representaciones de plantillas y estructuras de documentos que serán usados para producir textos semánticamente correctos. Para ello, el programa hará uso de un corpus previamente seleccionado y elaborado en base a artículos y descripciones de productos extraídos de Internet. En la siguientes secciones mostraremos cómo el material disponible se depura y procesa para elaborar recursos y algoritmos útiles para la generación de texto.

Nociones Preliminares

Para el desarrollo del trabajo que presentamos en este informe, es necesario estudiar y comprender varias herramientas y técnicas que usaremos a lo largo del proyecto. A continuación presentamos un resumen de los temas que abordaremos.

1.1. Generación de Lenguaje Natural

La Generación de Lenguaje Natural (GLN) es una área dentro del Procesamiento de Lenguaje Natural cuyo propósito es construir un texto en lenguaje natural para la comunicación con fines específicos. En términos más precisos, GLN consiste en usar representaciones computacionales para construir expresiones inteligibles en lenguaje natural de manera que se satisfaga un objetivo (u objetivos) comunicativo dado. Por lo general, un generador de texto tiene acceso a un gran conjunto de datos del cual ha de seleccionar información para presentar a los usuarios un texto legible y semánticamente fiel. Generar texto es, pues, un problema de toma de decisiones con múltiples restricciones: de conocimiento proposicional, de herramientas lingüísticas disponibles y de las intenciones de quien produce el texto. Se trata de identificar los factores involucrados en este proceso y de determinar la mejor forma de representar estos factores y sus dependencias.

Desde una perspectiva práctica, la GLN ofrece técnicas que se pueden aplicar para construir sistemas capaces de producir textos aceptables a partir de diversas fuentes. La GLN se usa para automatizar parcialmente las tareas de creación de documentos rutinarios, y para eliminar mucha de la carga monótona asociada a tales tareas. También se utiliza para presentar y explicar información compleja cuando el usuario no tiene la experiencia, la formación o el tiempo requerido para comprender los datos en bruto. Asimismo se emplea en otro tipo de sistemas, como la generación de informes meteorológicos, la producción de cartas de respuesta a clientes o la descripción de rutas. Finalmente, algunos sistemas experimentales han explorado otros usos de la GLN, generalmente ligados a la producción de textos creativos como cuentos, chistes y poesía.

1.1.1. Un poco de historia

Lo que conocemos como GLN nació entre los años 1950 y 1960 en proyectos de traducción automática. En la década del 60 también se pudo ver los primeros intentos de utilizar gramáticas de lenguaje natural como medio para la generación de sentencias bien formadas al azar. Los primeros trabajos dedicados exclusivamente a GLN comenzaron en los años 1970 con las investigaciones de Goldman ([7]) en que se intentaba elegir palabras apropiadas para expresar conceptos abstractos o las investigaciones de Davey interesadas en generar estructuras de textos y expresiones referenciales para la descripción de juegos. Estos intentos fueron pioneros y ayudaron a establecer algunas de las cuestiones principales en GLN. En los años 80 aparecieron un gran número de trabajos y tesis de doctorados que fueron muy influyentes para el campo de investigación. Entre ellos se incluyen los trabajos realizados por McKeown y Appelt en [6]. Durante la década de 1990 se produjo un crecimiento considerable, con muchas más objetivos comunicaciones, tesis doctorales, congresos y aplicaciones de investigación, y con la aparición de los primeros sistemas de GLN aplicados al mundo real. En la actualidad, la mayoría de los problemas que plantea la GLN están enunciados, pero no resueltos. Además, la GLN sigue siendo una labor cara y manual. La mayoría de los sistemas se construyen a medida y apenas se reutilizan recursos, por lo que generalmente cada proyecto de GLN implica elaborar técnicas y análisis *ad hoc*.

1.1.2. Arquitectura de sistemas de generación de lenguaje natural

A continuación mostramos ćomo es la arquitectura clásica de los sistemas de generación de lenguaje natural propuesta por Reiter y Dale en [4]. Estos autores proponen que un sistema de generación típico posee las siguientes entradas:

- la fuente: se refiere a las bases de datos del dominio y bases de conocimiento que contiene el material que proveen información para los textos que serán generados
- el objetivo de comunicación: especifica el propósito para el cual el texto es generado.
- el modelo de usario: provee información sobre el usuario como tareas, nivel de experiencia, preferencias, etc.
- la historia : información previa de interacciones entre el sujeto y el sistema.

En muchos sistemas de generación de lenguaje natural, la arquitectura se encuentra dividida en cuatro etapas:

- planificación de documento: determinar el contenido y la estructura de un documento

- microplanificación: decidir qué palabras, estructuras semánticas y demás serán usadas para comunicar el contenido y estructura elegida por el planificador de documento.
- realización: conecta las representaciones abstractas usadas por el microplanificador para convertirlas en texto concreto.
- presentacion física: diseño del texto (título, formato,etc)

Estos módulos están conectados en forma de tubería, es decir, la salida de cada etapa es la entrada de la siguiente.

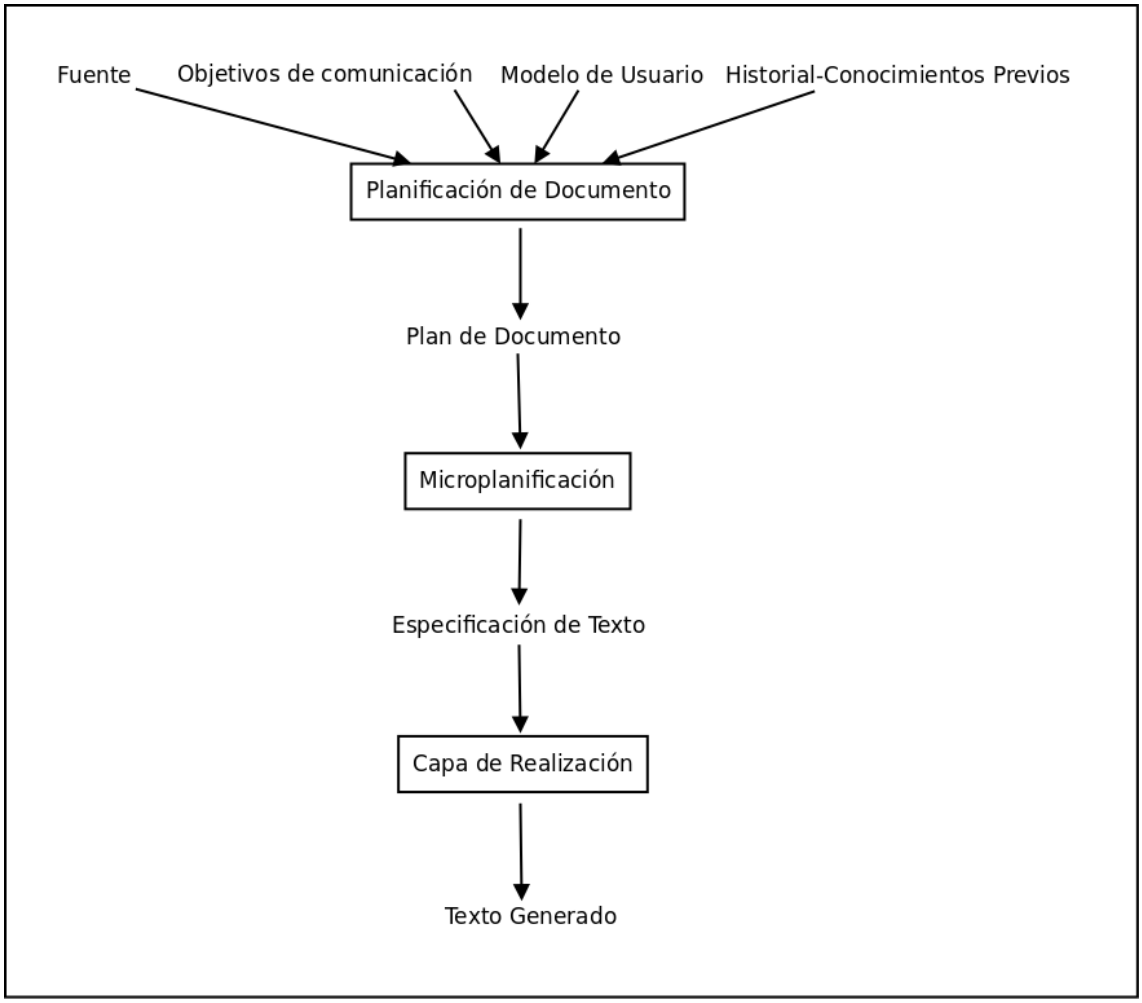

Figura 1.1: Arquitectura típica de los sistemas de Generación

A continuación mostramos algunos detalles generales de cada etapa.

Planificación de documento

El planificador de documento es responsable de decidir qué información comunicar(determinación de contenido) y determinar cómo esta información debe ser estructurada para ser presentada (estructuración de documento).
La determinación de contenido es la tarea de decidir qué información extraída de la fuente debe ser incluida en el texto generado, mientras que la estructuración de documento es el proceso que determina cómo se deben agrupar los elementos de contenido en un documento y cómo se deben relacionar esos elementos en términos retóricos.

Este proceso realiza tres tareas bien definidas:

- decidir qué partes de la información deben ser comunicados en orden de satisfacer los objetivos de comunicación
- realizar estructuras de documentos que organizan la presentación de la información de manera tal que se pueda generar texto coherente y fluído.

La salida de este proceso es un plan de documento.

Microplanificación

La tarea de este componente es tomar el plan de documento y refinarlo para producir una especificación de texto más detallada que pueda ser pasada a la etapa de generación. El plan de documento deja abierto un número de decisiones sobre la forma que tendrá finalmente el texto en el documento a ser generado. Estas decisiones son realizadas por este componente y son:

- Lexicalización: elegir las palabras, construcciones sintácticas y anotaciones usadas para comunicar.
- Agregación: decidir cuánta información debe ser comunicada en cada oración del documento. Se considera la agregación como el proceso de unir varios elementos de la información.
- Generación de expresiones de referencia: determinar qué frases deben ser usadas para identificar entidades particulares del dominio.

El resultado de este componente es una especificación de texto que contiene detalles más técnicos del documento a generar.

Realización

Una especificación de texto provee una completa especificación del documento a generar, pero no constituye por si sólo un documento. Es en esta etapa en que esa representación abstracta es mapeada a un texto real, es decir una secuencia de palabras con símbolos de puntuación. Este proceso concatena la información obtenida en las dos etapas anteriores en un texto gramaticalmente coherente.

1.2. Clustering

1.2.1. Definición

Clustering puede definirse como la tarea de agrupar un conjunto de datos de manera tal que que los objetos que forman parte de un grupo, llamado cluster, son similares (en algún sentido a definir) entre ellos. Es una técnica ampliamente usada en varios campos de estudio como minería de datos, procesamiento de lenguaje natural, machine learning y análisis estadístico entre otros.

1.2.2. Tipos de algoritmos de clustering

Hay una gran cantidad de algoritmos de clustering que pueden ser clasificados dentro de algunos tipos básicos. Según el tipo de estrucutura que generan los algoritmos, podemos encontrar dos tipos de clustering: clustering jerárquico y clustering plano o no jerárquico. El clustering plano simplemente consiste en un número de clusters y la relación entre ellos muchas veces no es determinada. La mayoría de los algoritmos que producen clusters planos son iterativos y comienzan con un conjunto inicial de clusters los cuales van mejorando en cada iteración utilizando una función de asignación de objetos a cada cluster. Un algoritmo de clustering jerárquico produce un árbol de jerarquía de clusters representados como nodos, en los que cada nodo representa una subclase del nodo padre. Las hojas del árbol son los objetos del conjunto de datos inicial. Cada nodo representa el cluster que contiene todos los objetos de sus descendientes.

Otra distinción importante entre los algoritmos de clustering es si realizan una asignación débil o fuerte. En una asignación fuerte cada objeto es asignado a un y sólo un cluster. Una asignación débil o suave permite que un objeto puede pertenecer a varios clusters. En el clustering jerárquico, la asignación es generalmente fuerte mientras que en el no jerárquico ambos tipos de asignaciones son usuales.

1.2.3. K-means

Definición

K-means es una técnica de clustering no jerárquico cuyo objetivo es clasificar o agrupar ítems dentro de k grupos (donde k es especificado por el usuario y determina la cantidad de clusters). El agrupamiento se hace mediante la minimización de la suma de distancias al cuadrado (distancia euclidiana) entre los ítems y el correspondiente *centroide*. Un centroide (también llamado vector medio) es el centro de masa de un objeto geométrico de densidad uniforme. Dado un cluster X Podemos definir al centroide como el promedio de los elementos que pertenecen a C.

$$centroide\ de\ X\ = \frac{1}{|X|} \sum_{x_i \in X} x_i$$

Aunque k-means es simple y puede ser utilizado para una gran cantidad de tipos de datos, es bastante sensible a las posiciones iniciales de los centros de los clusters. En general, hay dos métodos muy reconocidos para la inicialización de los centros: seleccionar los valores iniciales de manera aleatoria o elegir las k primeras muestras de los datos.

Este algoritmo selecciona arbitrariamente una cantidad K de objetos, cada uno de los cuales representa inicialmente un punto medio de un cluster o centro. Para cada uno de los restantes objetos, un objeto es asignado al cluster al cual es más similar, basada en la distancia entre el objeto y el punto medio del cluster. Luego computa un nuevo valor medio para cada cluster. Este proceso itera hasta que la función que se establece como criterio converge.

Características

Este algoritmo intenta determinar K particiones que minimizan los errores cuadráticos de la función de distancia. Es relativamente escalable y eficiente al procesar un gran conjunto de datos dado que la complejidad computacional del algoritmo es O(NKT) donde
N = cantidad total de objetos
K = cantidad de clusters
T = cantidad de iteraciones.

Parámetros de entrada y salida

El algoritmo toma como parámetros de entrada:
K cantidad de clusters
D conjunto de datos conteniendo n objetos
T cantidad máxima de iteraciones

y produce como salida:
Conjunto de K clusters.

Algoritmo

1. Arbitrariamente se eligen K objetos de los D iniciales como los centros de los clusters
2. Asignar cada objeto al cluster cuya distancia al centroide sea menor
3. Actualizar la media de los clusters: calcular el centroide de cada cluster
4. Repetir los últimos dos pasos hasta que no cambien los cluster o se alcancen T iteraciones

Distancia Euclideana

La función de distancia utilizada por el algoritmo es la distancia Euclidiana. La misma establece que la distancia en un plano de dos puntos con coordenadas (x, y) y (a, b) esta dada por:

$$distancia((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}$$

Debilidades del algoritmo

- con pocos datos, el agrupamiento inicial determina significativamente los clusters finales.
- el número de clusters K debe ser determinado de antemano
- con pocos datos puede ocurrir que los agrupamientos no sean precisos
- no podemos conocer qué variable contribuye más al proceso de clustering dado que asumimos que cada una tiene el mismo peso

- la precisión de la media matemática entre los objetos es fuertemente afectada por la presencia de datos atípicos, lo cual puede desplazar al centro de su posición.
- en las implementaciones usuales del algoritmo no es posible reasignar objetos en clusters ni tampoco dividir o mezclar clusters.

1.3. Part-of-Speech Tagging

Uno de los objetivos más importantes del procesamiento del lenguaje es comprender el mismo. Una de las tareas dedicadas a tal objetivo lo constituye el part-of-speech tagging o simplemente tagging. En español se conoce esta tarea como etiquetado gramatical y consiste en asignar una etiqueta a cada palabra en una oración indicando la categoría gramatical de ella. Este proceso se puede realizar utilizando la definición de la palabra u observando el contexto en el que ocurre.

1.3.1. Problemática

El etiquetado gramatical es una tarea compleja. Notemos que no es suficiente tener una lista de palabras con sus correspondientes categorías ya que muchas palabras pueden tener distintas categorías gramaticales según el contexto en donde ocurren.

Para dar respuesta a esta problemática podemos encontrar dos grandes grupos: aproximaciones lingüísticas y aproximaciones de aprendizaje automático. El primer grupo consiste de un conjunto de técnicas basadas en reglas establecidas de antemano por expertos. El segundo grupo comprende metodologías de aprendizaje automático que usan corpus de textos con información ligüística para establecer los modelos presentes en ellos.

A continuación mostramos algunos de los detalles más relevantes de las dos metodologías propuestas para solucionar la problemática.

Aproximaciones Lingüísticas

La principal ventaja de estas aproximaciones es que se construyen modelos de lenguaje desde un punto de vista lingüístico, por lo que se pueden incluir muchas y complejas fuentes de información, difíciles de capturar de manera automática. Este hecho las hace más expresivas, por lo que en general, suelen proporcionar mejores prestaciones en tareas de desambiguación si se comparan con otro tipo de aproximaciones. El problema de estos sistemas es que son totalmente dependientes de la lengua para la que se han sido diseñadas, y requieren un gran coste humano para la definición de las reglas.

Aproximaciones de Aprendizaje Automático

Estas aproximaciones construyen un modelo de lenguaje utilizando métodos de aprendizaje a partir de datos. Estas aproximaciones difieren entre sí en el método de aprendizaje y en la complejidad del modelo construido. Muchos son los formalismos utilizados: Modelos de Markov, n-gramas, reglas de transformación, árboles de decisión, redes neuronales, autómatas y transductores de estados finitos, etc. La aproximación más utilizada son los Modelos de Markov Ocultos.

Esta técnica consiste en construir un modelo de lenguaje estadístico, que se utiliza para obtener, a partir de una frase de entrada, la secuencia de etiquetados léxicos que tiene mayor probabilidad. Por ejemplo, si hemos etiquetado una palabra como artículo, la próxima palabra será un nombre con un 40 % de probabilidad, un adjetivo con otro 40 % y un número el 20 % restante. Conociendo esta información, un sistema puede decidir que la palabra "vino" en la frase "el vino" es más probable que sea un nombre a que sea un verbo.

Algunos MMO más avanzados aprenden las probabilidades de pares, triples e incluso secuencias más largas. Por ejemplo, si acabamos de etiquetar un artículo y un verbo, la siguiente palabra probablemente será una preposición, un artículo o un nombre, pero difícilmente será otro verbo.

Cuando aparecen varias palabras ambiguas juntas, las posibilidades se multiplican. Sin embargo, se puede calcular la probabilidad de cada secuencia de posibles etiquetados y seleccionar la secuencia de etiquetados con mayor probabilidad. El etiquetador CLAWS emplea este sistema y consigue un porcentaje de aciertos en el rango 93-95 %. Para más información véase [9].

1.3.2. TreeTagger

TreeTagger es un P.O.S Tagger elaborado por Helmut Schmid ([3]). Ha sido usado exitosamente para etiquetar textos en alemán, inglés, italiano, español y portugüés entre otros idiomas. Posee una precisión del 96.36 % y se diferencia de otros taggers probabilísticos utilizando árboles de decisión binarios para calcular las probabilidades de ocurrencias de etiquetas.

Para el idioma inglés, TreeTagger utiliza el conjunto de etiquetas definidas en el corpus Penn Treebank. En el cuadro 1.1 mostramos las etiquetas del corpus que son usadas más frecuentemente para etiquetar en Inglés.
Para ver la lista completa de etiquetas véase [18].

Etiqueta	Categoría gramatical
AT	artículo
BEZ	la palabra *is*
IN	preposición
JJ	adjetivo
JJR	adjectivo comparativo
MD	verbo modal
NN	sustantivo singular
NNP	nombre propio singular
NNS	sustantivo plural
PN	pronombre personal
RB	adverbio
RBR	adverbio comparativo
TO	la palabra *to*
VB	verbo infinitivo
VBD	verbo, tiempo pasado
VBG	verbo, presente participio, gerundio
VBN	verbo, pasado participio
VBP	verbo, tercera persona singular presente
VBZ	verbo, tercera persona singular presente
WDT	determinante $wh-$ ($what$,$which$)

Cuadro 1.1: Etiquetas más usadas para tagging en inglés

1.4. Introducción a SVM

En el capítulo 7, usaremos una técnica de aprendizaje automático llamada Support Vector Machine. Aquí detallamos brevemente algunos conceptos básicos de esta técnica y características generales de la misma.

1.4.1. Definición

En Aprendizaje Automático, las Máquinas de vectores de soporte (SVMs) son modelos de aprendizaje supervisado que comprenden un conjunto de algoritmos de aprendizaje que analizan datos y reconocen patrones, usados para los análisis de clasificación y regresión. El algoritmo original de SVM fue inventado por Vladimir N. Vapnik y Alexey Ya. Chervonenkis en 1963. Las versiones estándares más actuales fueron propuestas por Corinna Cortes y Vapnik en 1993 y publicados en 1995.
Dado un conjunto de ejemplos de entrenamiento, en los que cada uno tiene asignado una categoría o dos, un algoritmo de SVM elabora un modelo que permite asignar nuevos ejemplos dentro de una categoría u otra.

1.4.2. Idea general de un modelo de SVM

Un modelo de SVM puede ser pensado como una representación de los ejemplos como puntos en un espacio de manera que aquellos puntos que pertenecen a distintas categorías son separados por un margen lo más amplio posible. Para los nuevos ejemplos se calcula la distancia a estos márgenes y de esta manera se determina a qué categoría deberia pertenecer.

Más formalmente un SVM construye hiperplanos o un conjunto de ellos en un espacio de alta dimensionalidad. SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra. Intuitivamente una buena separación es lograda por el hiperplano que tiene la mayor distancia respecto del ejemplo de entrenamiento más cercano.

1.4.3. Características

En este concepto de separación óptima es donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. Por eso también a veces se les conoce a las SVM como clasificadores de margen máximo. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado. Un punto que representa un ejemplo de entrenamiento, es visto como un vector p-dimensional (es decir, una lista de p números) y se intenta conocer si los distintos puntos pueden ser separados por un hiperplano (p-1)-dimensional. Este tipo de SVM es llamado clasificador lineal. A los vectores formados por los puntos más cercanos al hiperplano se los llama vectores de soporte. En la Figura 1.2 mostramos un ejemplo de clasificación de muestras en dos dimensiones y algunos hiperplanos que se generan para dividir a la misma: H_1, H_2 y H_3. Para más información véase [13].

1.5. Distancia de Levenshtein

Esta distancia, llamada así por su inventor Vladimir Levenshtein ([10]), es el número mínimo de operaciones que se necesitan para convertir una cadena de caracteres o string en otra. Dada una cadena de caracteres c de longitud len , una operacion sobre esta cadena puede ser de tres tipos:

- inserción: Agregar un nuevo caracter h en la posición pos con $0 \leq pos \leq len$.

$$c = c[0...(pos-1)] + h + c[pos..len]$$

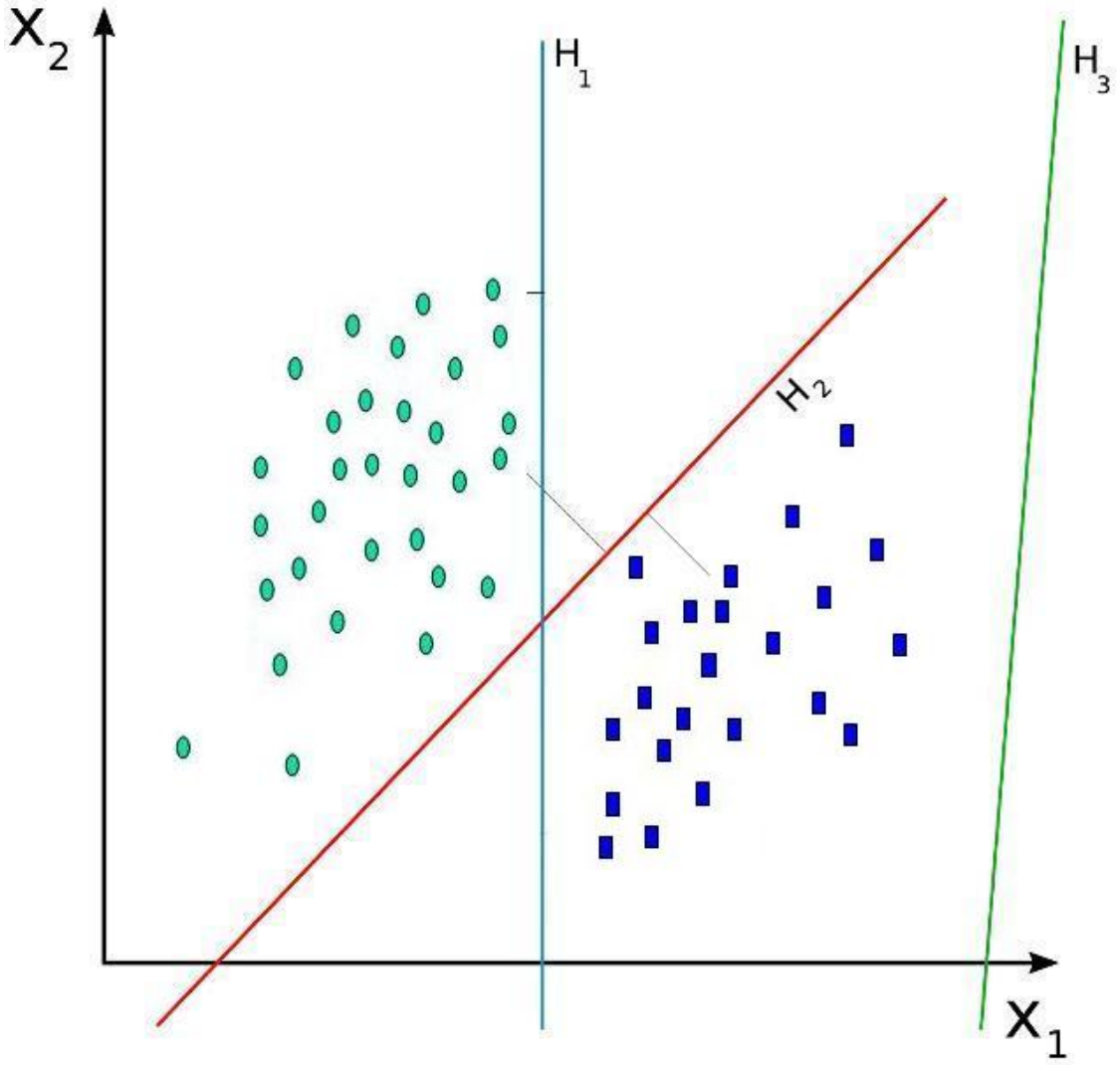

Figura 1.2: Ejemplo de datos agrupados por SVM

- eliminación: Eliminar el caracter de la posición pos con $0 \leq pos \leq len$.

$$c = c[0...(pos-1)] + c[(pos+1)..len]$$

- sustitución: Cambiar el caracter de la posición pos por el caracter h con $0 \leq pos \leq len$.

$$c = c[0...(pos-1)] + h + c[(pos+1)..len]$$

Se le considera una generalización de la distancia de Hamming, que se usa para cadenas de la misma longitud y que sólo considera como operación la sustitución. Hay otras generalizaciones de la distancia de Levenshtein, como la distancia de Damerau-Levenshtein, que consideran el intercambio de dos caracteres como una operación. Es ampliamente usada en teoría de la información y ciencias de la computación. Para citar un ejemplo

observemos que la distancia Levenshtein entre *casa* y *calle* es 3, ya que se necesitan tres operaciones para cambiar una por otra:
casa -> cala (sustitución de s por l)
cala -> calla (inserción de l entre l y a)
calla -> calle (sustitución de a por e)

1.5.1. Algoritmo

A continuacioń mostramos el algoritmo en pseudocódigo para implementar la función LevenshteinDistance. Toma como parámetros dos cadenas str1 de longitud lenstr1 y str2 de longitud lenstr2 y computa la distancia entre ellos.

```
1  int LevenshteinDistance(char str1[1..lenStr1], char str2[1..lenStr2])
2      // d es una tabla con lenstr1 + 1 filas y lenstr2 + 1 columnas
3      int d[0..lenStr1, 0..lenStr2]
4
5      int i, j, cost
6
7      for i from 0 to lenStr1
8          d[i, 0] := i
9      for j from 0 to lenStr2
10         d[0, j] := j
11
12     for i from 1 to lenStr1
13         for j from 1 to lenStr2
14             if str1[i] = str2[j] then cost := 0
15                                  else cost := 1
16             d[i, j] := minimum(
17                                  d[i-1, j] + 1,      // eliminacion
18                                  d[i, j-1] + 1,      // inserci\'on
19                                  d[i-1, j-1] + cost   // sustituci\'on
20                                )
21
22     return d[lenStr1, lenStr2]
```

1.6. Precisión y Cobertura

Para analiizar una prueba de reconocimiento de patrones con clasificación binaria existen estadísticos y test clásicos asociados para determinar la validación del experimento. Precisión y Cobertura es una métrica empleada en la medida del rendimiento de los sistemas de búsqueda y recuperación de información y reconocimiento de patrones.
Estas métricas pueden ser expresadas en términos de instancias que el sistema a validar detecta o no. Podemos clasificar cada instancias en tres grupos distintos:

- Tp, True Positive: el sistema lo detectó y existía. (Acierto)
- Fp, False Positive: el sistema lo detectó y no existía. (Confusión)
- Fn, False Negative: el sistema no lo detectó y existía. (Omisión)

Utilizando estas clasificaciones podemos reescribir las métricas de la siguiente manera:

$$Precision = \frac{Tp}{Tp + Fp}$$

$$Recall = \frac{Tp}{Tp + Fn}$$

La situación ideal es aquella en la que existe una precisión y exhaustividad alta (es decir muy cercana a 1). A esta situación se la denomina utilidad teórica. Con el objeto de ponderar y ver cuán lejano se encuentran ambas medidas de la utilidad teórica, suele emplearse los valores de ambas métricas combinadas en una media armónica denominada valor-F la cual indica la medida de precisión que tiene un test.

$$F = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)}$$

■

Sobre el corpus

2.1. Extracción/origen

Para realizar nuestro proyecto y los experimentos relacionados, contamos con corpus previamente seleccionado de avisos clasificados. Éstos fueron extraídos de Internet de manera automática utilizando un script escrito en Php. Éste es un crawler o araña Web cuya funcionalidad es la de inspeccionar dominios de Internet dedicados a la publicación de clasificados.
El script fue desarrollado de manera tal que sea capaz de reconocer la ubicación de textos relacionados a avisos dentro de la página Web que utilizamos como fuente. Generalmente, las páginas Web poseen una estructura fija para su contenido determinando qué etiquetas se utilizan para cada tipo de información como tablas, imágenes y texto. Esta estructura tiene el modelo de árbol de etiquetas y fue lo que aprovechamos para construir el algoritmo que detecte avisos. Para esta tarea contamos con un Modelo de Objetos de Documentos (DOM) de HTML, que es una Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) que define la estructura lógica de los documentos y la manera en que se acceden y manipulan. Con esta herramienta, logramos recorrer una entidad HTML iterando sobre los tags que contiene y extraer la información.
Para entender como es el funcionamiento básico del crawler, consideremos un dominio Web que aloja sus avisos clasificados en un entidad HTML que tiene la siguiente estructura:

<TABLE>
<TBODY>
<TR>
<TD>Título del aviso 1</TD>
<TD>Texto del aviso 1</TD>
</TR>
<TR>
<TD>Título del aviso 2</TD>
<TD>Texto del aviso 2</TD>
</TR>
</TBODY>
</TABLE>

Nuestro DOM crea una representación de esta tabla en forma de árbol.

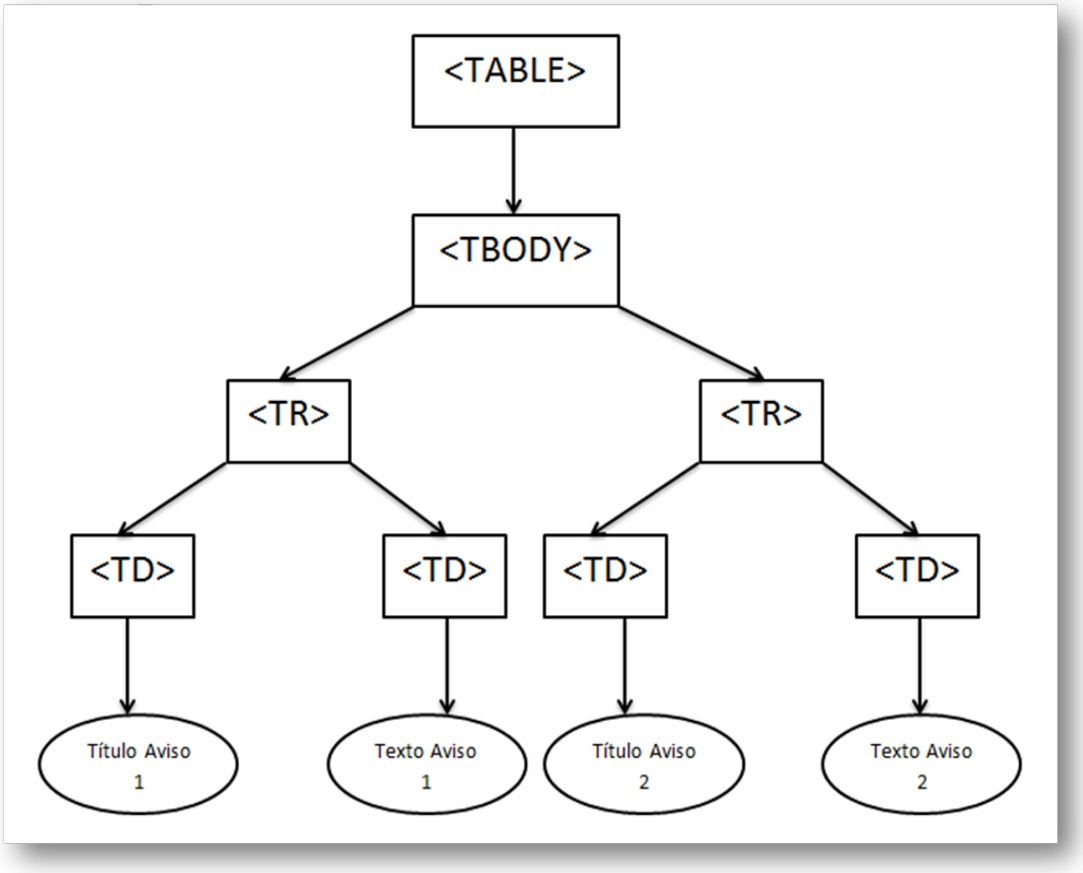

Figura 2.3: Representación generada por el DOM

Iterando sobre el árbol, podemos acceder a las hojas de éste y de esta manera obtener los avisos contenidos en la tabla. El siguiente es un ejemplo en pseudocódigo del algoritmo que puede ser utilizado para recorrer la tabla y extraer título y texto del aviso.

```
1  \\ entrada: tabla html
2
3  avisos = arreglo vacio
4  por cada TABLE
5    por cada TBODY
6      por cada TR
7        titulo = primer TD
8        texto = segundo TD
9         push(avisos, (titulo,texto))
10
11 return avisos
```

Este ejemplo resume el funcionamiento del script que desarrollamos para obtener nuestro corpus. Sólo bastó con identificar la estructura que utiliza nuestra fuente para alojar

los avisos y elaborar el algorimo *ad hoc* para iterar sobre el árbol de etiquetas creadas por el DOM para esa estructura.

Con el fin de acotar nuestro dominio de trabajo, se seleccionaron sólo un grupo de avisos que presentaban una estructura muy específica. Ésta fue determinada para representar de manera general la forma más básica (según nuestra apreciación) de anuncio disponible o visible en Internet. Dicha estructura presenta características propias de avisos clasificados publicados en sitios de compraventa. Es importante aclarar que en numerosos sitios la publicación de cierto aviso queda limitado a un número fijo de caracteres y el usuario debe ser capaz de expresar de manera concreta y completa la información que dispone sobre su producto.

2.2. Estructura

A continuación se muestra la estructura general que presentan las publicaciones que componen el corpus:

$$\langle aviso \rangle ::= \langle encabezado \rangle \textbf{ : } \langle detalle \rangle$$

Podemos distinguir dos secciones bien definidas: el *encabezado* y el *detalle*. El primero es un segmento del anuncio sumamente importante para el lector o interesado: el mismo debe resumir brevemente el tipo de producto a vender y características interesantes o destacadas sobre el mismo. Podemos representar este componente mediante la siguiente gramática:

$$\langle encabezado \rangle ::= \langle descr \rangle \textbf{ - } \langle precio \rangle \textbf{ (}\langle lugar \rangle\textbf{)}$$

donde:

- $\langle descr \rangle := \langle string \rangle$

  Es un string que detalla alguna característica destacada del anuncio.

- $\langle precio \rangle := \textbf{\$}\langle num \rangle \mid \langle num \rangle\textbf{\$}$

  Representa el valor del producto/servicio a vender

- $\langle lugar \rangle := \langle string \rangle$

  String que determina el lugar desde donde se realiza la oferta o el lugar donde se ubica el producto.

Por otra parte, el detalle es la sección que contiene una descripción bastante más amplia o completa del artículo. En ella podemos encontrar generalmente descripciones del objeto a vender que hacen referencia al tamaño, color, estilo, estado, nacionalidad, etc. Aquí también se suele detallar información de contacto y forma de pago/entrega.

2.3. Composición

Nuestro material de estudio seleccionado está compuesto de 9498 artículos representados como pares del tipo dado en la sección anterior. Dentro del corpus podemos identificar siete categorías que representan el rubro al cual corresponde cada aviso. La idea de esta sub-clasificación es estudiar la variabilidad del texto descriptivo según las diversas categorías: El objeto que se está comercializando determina de manera contundente la forma de expresar las características y los detalles importantes del mismo. Este hecho es claramente visible ya que es fundamental para lograr el interés en el lector/consumidor, por lo que la persona encargada de publicar tiene presente detalles como la fecha de fabricación en el caso de que el objeto sea una antigüedad, mientras que por otro lado debe destacar el color y tamaño si se tratase de un mueble para el hogar. Los mencionados rubros son: antigüedades, muebles, productos de belleza, productos de colección, productos de granja, productos del hogar y joyas. En el Cuadro 2.2 vemos cómo se encuentran distribuídos los anuncios según el rubro.

Rubro	**Cantidad de avisos**	**Cantidad de oraciones**
antigüedades	339	1409
muebles	144	629
productos de belleza	484	1806
productos de colección	360	1465
productos de granja	405	1870
productos del hogar	419	1543
joyas	162	776

Cuadro 2.2: Distribución de avisos según rubro.

2.4. Limpieza/Problemas usuales del corpus

Debido a la naturaleza de la información, es posible distinguir numerosos problemas en las tareas de pulir y adecuar el corpus para su análisis. Entre los inconvenientes podemos mencionar la presencia de etiquetas HTML, caracteres especiales y errores de redacción en general.

2.4.1. Depuración/HTML

El corpus fue seleccionado desde Internet como se mencionó previamente. Los textos que fueron extraídos son entidades de tipo HTML, por lo que se requirió cierto trabajo especial para convertir tal contenido en texto plano.

El principal inconveniente recae en la presencia de tags propios de HTML. En general, este problema fue resuelto gracias a la utilización de funciones específicas de Php cuya finalidad es eliminar tags del lenguaje HTML. Sin embargo, podemos ver que la tarea no es tan sencilla ya que no es posible distinguir de forma precisa la ubicación de las imágenes que acompañan a los anuncios en el texto. Nuevamente mediante Php es posible utilizar funciones que permiten reemplazar texto por medio del uso de expresiones regulares. Aprovechando el poder expresivo de las expresiones regulares se procedió a limpiar las oraciones de imágenes, links y eliminar algunos caracteres propios de HTML y otros ajenos a la codificación UTF-8. Por otro lado, además de la transformación del texto HTML a texto plano, se confeccionó un script que eliminaba las oraciones y párrafos repetidos del corpus.

2.4.2. Errores de redacción

Los textos seleccionados presentan numerosos errores de redacción los cuales no fueron solucionados completamente: entre estos se encuentran la presencia de palabras mal formadas o mal escritas y la falta o abuso de signos de puntuación. Sin bien utilizamos correctores ortográficos para intentar arreglas las palabras mal escritas, en general estos problemas no fueron corregidos en su totalidad dado que se requería la verificación manual de todos los textos.

2.4.3. Separar oraciones

En la etapa final de este proceso de limpieza, se dispuso a separar el texto en su totalidad en oraciones. Tal como se mencionó, Php dispone de numerosas funciones que permiten procesar el texto por medio de expresiones regulares. Ésto fue clave a la hora

de transformar este proceso en un tarea semi automática. Como primera medida se optó por identificar las oraciones terminadas con el carácter "." . Aquí fue necesario recurrir a las expresiones regulares para identificar aquellas presencias de signos de puntuación que no representaban un fin de oración. En esta categoría se encuentran aquellos caracteres utilizados para determinar precios,dimensiones, números en general, abreviaciones o algún recurso expresivo. Con parte de la tarea resuelta por medio de scripts y funciones desarrollados en Php, se inspeccionó manualmente el corpus con el fin de encontrar aquellas excepciones no consideradas previamente.

■

Arquitectura del Sistema

En general los sistemas de generación de texto utilizan metodologías basadas en reglas donde un conjunto de datos es manipulado para generar texto. Esta reglas existen en diferentes niveles del sistema ya sea desde la selección de contenido hasta definir la forma gramatical de la salida del mismo. Las decisiones tomadas en cada etapa del sistema son realizadas por reglas definidas ya sea por expertos o usando trabajos lingüísticos previos. Por otro lado, existen también sistemas de generación estadísticos donde la construcción de estas reglas son aprendidas estadísticamente mediante un análisis de la fuente de información con distintos procesos automáticos.

Nuestro sistema fue diseñado combinando dos metodologías: aproximaciones basadas en templates y aproximaciones estadísticas. Primero aprendemos estructuras semánticas del corpus que son usadas luego para producir templates para nuestro sistema (combinando las etapas de Microplanificación y Realización). Luego creamos un modelo estadístico para seleccionar los mejores templates que usaremos para generar. Esta combinación permite minimizar los procesos manuales de verificación y además es muy adaptable para diferentes dominios. Este sistema fue diseñado siguiendo el modelo de trabajo propuesto en [12]. Con intención de clarificar los pasos ejecutados en el proyecto y guiar al lector durante las distintas secciones, presentamos la arquitectura general del sistema que puede ser resumida en cinco etapas.

A - **Representación semántica** Tomamos un corpus compuesto de avisos clasificados y obtenemos dos representaciones semánticas del mismo. Creamos un algoritmo para identificar etiquetas específicas del dominio llamado **AdTagger** mediante el cual extraemos información y datos relevantes de los avisos que nos ayudaran a crear un banco de templates.

B - **Clustering** Una vez que obtenemos la representacion semántica, agrupamos las oraciones utilizando K-means con K suficientemente grande y asociamos las oraciones y los templates generados a cada cluster. Esta etapa se corresponderá con el aspecto de Microplanificación de nuestro sistema.

C - **Estadísticas** En esta etapa recolectamos una serie de estadísticas que nos ayudarán a construir un modelo probabilístico.

D - **Ranking model** Creamos un modelo con el fin de determinar las probabilidades asociadas a cada template. La función principal de esta etapa es poder evaluar y determinar qué template es el más adecuado para utilizar en la etapa de generación. Representa la etapa de Planificación de Documentos de nuestro sistema.

E - **Generación** En esta etapa final, contamos con el modelo para establecer preferencias entre los templates del inciso anterior. La tarea de generación se resume en los siguientes pasos:

- Tomamos un aviso y lo dividimos en oraciones. Extraemos además los datos del mismo.
- Por cada oración buscamos los templates que corresponden en número y clase de etiquetas.
- Para cada template recolectamos un número de estadísticas.
- Utilizando el modelo obtenemos el template más significativo para cada oración.
- Por último llenamos esos templates con los datos de cada oración.

■

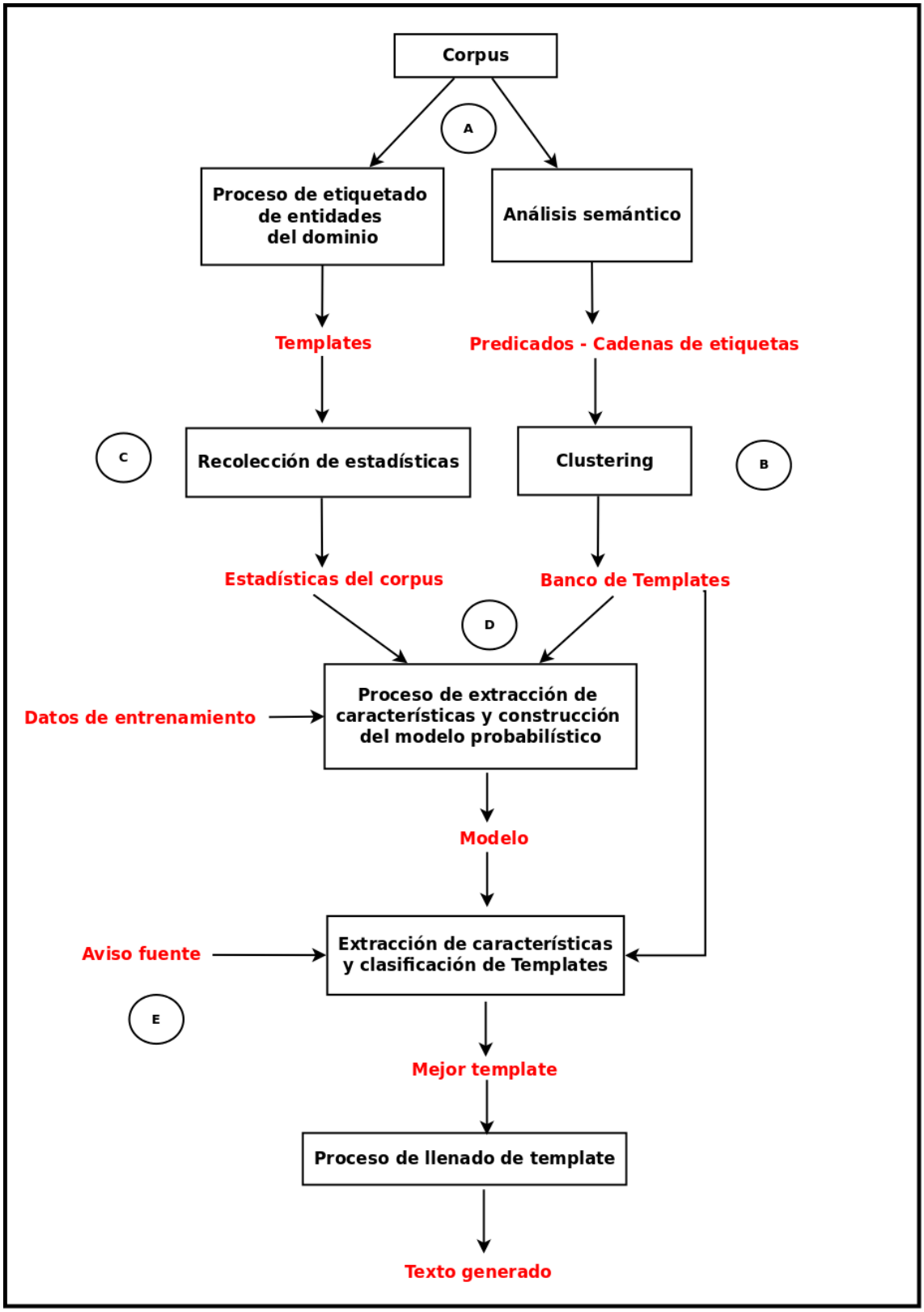

Figura 3.4: Arquitectura del sistema

AdTagger

Uno de los componentes claves para el funcionamiento de este sistema de generación automática lo conforma el etiquetador de entidades específico del dominio. Para realizar las distintas actividades era indispensable el uso de algún algoritmo entrenado en el dominio conformado por avisos clasificados. Para tal fin, se desarrolló un algoritmo específico del dominio (restringido al corpus) capaz de indentificar tags semánticos dentro de las oraciones. A continuación se detalla el desarrollo de AdTagger, sus principales componentes, funcionamiento y características particulares.

4.1. Desarrollo

Para desarrollar tal algoritmo se discutieron diversos métodos para encontrar etiquetas apropiadas para el dominio de los avisos clasificados. Finalmente nuestra propuesta fue encarar el problema de la identificacion de tags utilizando taggers sintácticos y otras herramientas de análisis semántico de palabras ya desarrollados y disponibles. En este caso nos valimos de dos herramientas bastante conocidas en el ámbito del procesamiento del lenguaje natural. Ellas son TreeTagger y Wordnet.

La tarea se dividió en diversas etapas y cada una de ellas requirió desarrollar algoritmos específicos.

4.1.1. Identificación de tags

Como se dijo en algún capítulo anterior, un aviso clasificado transmite alguna información respecto al objeto o servicio que se intenta publicitar. Debido a la variedad de rubros, esta información varía considerablemente respecto de las características y detalles. Parte de esta información hace referencia directa al objeto y otra hace referencia a propiedades del aviso, como precio, lugar,etc.

A modo de ejemplo consideremos el siguiente aviso:

Ejemplo 1
“ Antique chair - $350 (pacific heights) : Antique victorian Chair from the late 1800’s, Excellent condition. Call me if you’re interested! ”

En este ejemplo particular podemos identificar componentes que describen al objeto y componentes auxiliares que ayudan a constituir al aviso en sí. Distinguimos palabras que podemos clasificar y/o agrupar dentro de alguna palabra más general. Estas palabras son las denominadas hiperónimos. En el ejemplo vemos que “chair” es el objeto que se intenta vender. Para describir al objeto se utilizan palabras como “antique”, “victorian” o frases como “from the late 1800’s” y “ excellent condition”.
Estas brindan abundante información que se puede clasificar como:

estilo : antique, victorian
origen, tiempo : late 1800’s
estado del objeto, condición : excellent

Por otra parte existen en el aviso otros componentes con otra información relevante como precio, lugar de venta, etc. En este caso podemos distinguir entonces otras entidades como:

precio : $350
lugar : pacific heights
tipo de contacto : call

Ejemplo 2
“ Garden Table - $250 (burlingame) : This is a beautiful table that is green has a interesting design. It is solid wood. It is even more lovely in person. Please e-mail me for more information. ”

En este ejemplo observamos nuevos componentes que requieren otras clasificaciones dinstintas a las del ejemplo anterior. Estas son:

color : green
material : wood

Luego de un análisis exhaustivo de muestras del corpus, se pudo identificar diversas clasificaciones de la información que brindan los anuncios y se seleccionó un grupo de ellos para los experimentos. El tagger no supervisado se construyó de manera tal de poder

identificar y clasificar componentes que recaen en un grupo de esas seleccionadas. En el Cuadro 4.3 vemos cuáles son y qué representan.

Etiqueta	**Significado**
estilo	forma,estructura del objeto
tamaño	tamaño del objeto
precio	precio al cual se comercializa el objeto
color	color del objeto
nacionalidad	adjetivo gentilicio/ nacionalidad
material	sustancia de la que se compone el objeto
tiempo	referencia a algun periodo
estado	situacion actual,adjetivo calificativo que hace referencia estado
dimensión	longitudes,dimensiones expresadas en numeros
forma de pago	tipo de pago acordado
tipo de contacto	forma de contactar al anunciante
nombre	nombre propio presente en el aviso
caracteristica	detalles extras del aviso
lugar/origen	lugar donde se encuentra el objeto/anunciante
sexo	indica género

Cuadro 4.3: Etiquetas y sus significados

4.1.2. Análisis sintáctico de oraciones con TreeTagger

Una vez definida la información que se considera importante para extraer, se procedió a analizar una a una las oraciones en busca de palabras o frases que pertenezcan a las clasificaciones presentadas.

La idea principal de esta etapa era poder distinguir las clasificaciones sintácticas de palabras y con ello separar del texto aquellos vocablos que nos resultan útiles para distribuirlas en categorías semánticas.

Para desarrollar este análisis sintáctico, se utilizó TreeTagger. Esta herramienta nos proporciona información sintáctica útil sobre los componentes.
Para ejemplificar, continuemos con el ejemplo (1) y veamos cómo TreeTagger analiza el cuerpo del aviso:

Antique JJ antique
victorian JJ Victorian
Chair NN chairv
from IN from
the DT the
late RB late
finished.
1800s JJ <unknown>
Excellent JJ excellent
condition NN condition
. SENT .
Call VB call
me PP me
if IN if
youre NN <unknown>
interested JJ interested
! SENT !

Podemos observar que la herramienta clasifica en categorías sintácticas cada palabra que compone la oración y nos da el lema que sirve para Wordnet. De estos datos nos interesan aquellos que indican cuál es el objeto o que determinan alguna característica del mismo. Cabe destacar que el programa en ocasiones clasifica composiciones de palabras entre sí o palabras y caracteres. De este modo composiciones como "Homer Simpson" y "brand-new" son etiquetadas juntas con un solo tag.

De este modo aquellos términos marcados como adjetivos (JJ) o sustantivos (NN) fueron extraídos por medio de un script y colocados en archivos separados, uno para cada tipo de palabra.

Para garantizar que los datos extraídos sean correctos, es decir, que ningún vocablo que haya sido mal categorizado por TreeTagger sea incorporado a la lista de términos válidos, se procedió a realizar un depuración manual de los archivos generados.

Para realizar este proceso de análisis y depuración se ejecutó el programa para cada oración del corpus. Sin embargo para extraer los sustantivos, se procedió a dividir el corpus en siete archivos, uno por cada rubro y se repitió el experimento para cada fragmento por separado. Este modus operandi no es un hecho aislado y tiene su razón de ser en una particularidad de los idiomas en general: una palabra tiene diversas acepciones y es el contexto en el cual se usa el que determina el significado con el que debe interpretarse.

Veamos un ejemplo de este hecho. Consideremos la palabra "link" y consideremos los siguientes dos anuncios que corresponden a los rubros Productos de granja y Antiguedades respectivamente :

Anuncio 1:

“ Chain Link Fence Gate - $100 (healdsburg / windsor) : Chain Link Fence Gate, approx 8-1/2’ wide and 6’ tall. ”

Anuncio 2:

“ Antique ceramic desk lamp - $25 (berkeley) : Nice antique ceramic desk lamp with bulb. You can see it on this link example.com. ”

En este ejemplo, podemos observar como el significado de la palabra “link” puede variar según el contexto en el que está usado y cobrar o no importancia para nuestro tagger. Observemos que en el anuncio correspondiente al rubro de Productos de granja, la palabra forma parte del objeto del anuncio, mientras que en el segundo ejemplo no tiene ningún tipo de relevancia para el objeto.

Debido a este hecho, los sustantivos fueron extraídos y almacenados de forma separada para cada rubro para evitar caer en errores de interpretación.

4.1.3. Análisis de hiperónimos con Wordnet

A esta altura del desarrollo se disponía de una gran cantidad de términos extraídos del texto que eran candidatos a formar parte de alguna categoría semántica propuesta. El problema ahora radicaba en encontrar un método que ayude a determinar de manera efectiva qué palabras se identifican con una categoría dada.
Este problema fue abordado mediante el uso de la herramienta Wordnet. La misma contiene una enorme base de datos de palabras asociadas con sus hiperónimos. Con esta información presente, se desarrolló un procedimiento para dar solución al problema.
Para comprender cómo es el procedimiento propuesto, veamos como es una salida típica de la herramienta. Consideremos la palabra “blue”, el correspondiente análisis es el siguiente:

blue, blueness
=> chromatic color, chromatic colour, spectral color, spectral colour
=> color, colour, coloring, colouring
=> visual property
=> property
=> attribute
=> abstraction, abstract entity
=> entity

Dado que Wordnet contiene un extenso diccionario de acepciones para la palabra, sólo se muestra en el ejemplo el primer significado. Podemos observar aquí que hay un

conjunto de hiperónimos que, seleccionados, pueden ser propios de un color. En este caso por ejemplo podemos tomar la 3-upla chromatic color, chromatic colour, spectral color y ver que se repite en otro color como "yellow":

yellow, yellowness
=> chromatic color, chromatic colour, spectral color, spectral colour
=> color, colour, coloring, colouring
=> visual property
=> property
=> attribute
=> abstraction, abstract entity
=> entity

Una vez que observamos estos detalles,elaboramos un procedimiento. El mismo consta de los siguientes pasos:

1. Análisis de hiperónimos de Wordnet:
   Utilizando un conjunto de palabras seleccionadas manualmente,y que forman parte de una categoría dada, se procede a evaluar la salida de Wordnet. De este paso obtenemos un listado extenso de hiperónimos que maneja la aplicación.

2. Elaboración de diccionarios de hiperónimos que identifiquen una categoría.
   Una vez que tenemos disponibles los datos extraídos del paso anterior, construimos automáticamente diccionarios de hiperónimos que representen unívocamente a una categoría.

3. Análisis de las palabras con Wordnet y determinación de categoría
   Por último, para averiguar si una palabra pertenece a una categoría, se evalúa con Wordnet y se analiza si el patrón de hiperónimos representantes de una categoría está presente en la salida del programa.

Si bien este procedimiento fue útil para la gran mayoría de los vocablos extraídos en el paso anterior, había términos que no aparecían en Wordnet y por lo tanto, éstos debieron ser clasificados manualmente.
Otro aspecto importante para destacar recae sobre la política tomada cuando una palabra cae en dos o más categorías distintas. En este caso se dispusieron las siguientes medidas:

a) Se separa el vocablo y se enumeran las categorías

b) Se evalúa en qué acepción de la palabra cae cada categoría. La categoría con la acepción más cercana a la palabra es la elegida.
   Si aún hay dos o más categorías que cumplen esta propiedad, pasar al paso c.

c) Se evalúa qué categoría está más arriba en el nivel de hiperónimos. Si aún no se puede determinar la categoría entonces se anexa la palabra a un archivo de indeterminados para proceder a una verificación manual.

4.1.4. Almacenamiento y creación de gazetteers

Una vez completas las etapas de análisis, se obtuvo entonces un conjunto de palabras filtradas según las categorías que se plantearon. El siguiente paso fue la creación de gazetteers, uno por cada clase.
En cada uno de estos se almacenaron entonces los vocablos y términos extraídos de las etapas anteriores.

La distribución de palabras por gazetteer puede ser resumida en el Cuadro 4.4:

gazetter	**Cantidad**
estilo	49
tamaño	40
color	40
nacionalidad	68
material	118
tiempo	122
estado	132
forma de pago	6
tipo de contacto	8
nombre	69
característica	298
lugar/origen	67
sexo	2

Cuadro 4.4: Distribución de palabras según gazetter

Como podemos observar, hemos omitido en el cuadro las etiquetas de precio y dimensión. Eso se debe a que tales categorías son identificadas mediante el uso de expresiones regulares, sin necesidad de crear gazetteers para ellos. Por otro lado, la información de las etiquetas de tiempo y lugar/origen que se muestran en el cuadro corresponden sólo a las palabras contenidas en sus respectivos gazetteers. Sin embargo hemos usado también expresiones regulares para ampliar la cantidad de entidades a reconocer en el texto.

4.1.5. Creación del sistema de reemplazo

A esta altura del desarrollo, se contaba con la suficiente información para elaborar un algoritmo que reemplace en el texto las palabras por sus correspondientes tags. Esta última tarea requería especial atención para no etiquetar erróneamente vocablos y garantizar que las palabras de interés dentro del texto sean etiquetadas correctamente.

A primera vista, el algoritmo puede parecer sencillo de realizar: simplemente buscar en el texto las entidades que almacenamos en los gazetteers y cambiarlas por su correspondiente tag. Sin embargo es necesario hilar un poco más fino a la hora de evitar errores en el algoritmo. El primer problema que se presenta la hora de reemplazar texto, es evitar que aquellos strings que son substrings de otros sean etiquetados.

Por ejemplo, consideremos la palabra "new" del gazetteer que contiene las palabras relacionadas al estado del objeto. Uno pretendería que tal palabra en la frase "The paint has been renewed a few time ago" sea ignorada, puesto que solo aparece como substring de la palabra "renewed".
Para garantizar tal funcionamiento, se ideó una rutina que encuentra la ocurrencia de un término a buscar que no sea substring de algún otro. Esta rutina se basa fundamentalmente en la creación de expresiones regulares que identifican la palabra a buscar junto con un contexto mediante el cual se puede evaluar si se encuentra o no contenida en otra palabra.
La idea es la siguiente: buscar la palabra e identificar un carácter antes de la ocurrencia de la misma y un caracter posterior a la presencia de ella. Utilicemos nuevamente la palabra del ejemplo anterior para entender el funcionamiento. Para buscar "new" en la oración mencionada previamente, se crea una expresión regular que contenga a la misma junto con dos caracteres auxiliares al principio y fin, esto es, confeccionamos la expresión regular:
$(\neg(a-z))\text{new}(\neg(a-z))$
Esta expresión regular garantiza que la palabra "new" no esté rodeada de caracteres alfabéticos(lo que supondría que está contenida en otra). De esta manera, podemos identificar de manera concreta el string a reemplazar y lo cambiamos por su correspondiente tag, preservando los caracteres contextuales.
La rutina puede resumirse entonces en los siguientes pasos:

1. Dada la palabra WORD, construir la expresión regular $(\neg(a-z))\text{WORD}(\neg(a-z))$
2. Alamacenar en cStart y cEnd los caracteres de comienzo y fin extraídos de la expresión regular anterior
3. Reemplazar $(\neg(a-z))\text{WORD}(\neg(a-z))$ por (cStart)TAG(cEnd).

A este rutina, se le agrega además la verificación de que la palabra a buscar se encuentre al comienzo de la oración(pues en ese caso no cumple con la expresión regular ya que no posee caracter anterior. El caso de que sea fin de oración no es tenido en cuenta ya que

siempre se puede encontrar un carácter de fin como "." o "\n"). Para consultar más sobre expresiones regulares en Php véase [16]

4.1.6. Subetiquetadores

Una vez desarrollado el sistema para reemplazar palabras por etiquetas, nos dispusimos a crear las subrutinas encargadas de buscar y etiquetar las palabras almacenadas en cada gazetteer. Dado que la metodología para reemplazar palabras es igual para todos los gazetteers y las palabras contenidas en ellos, decidimos elaborar una función general llamada $findTags$ que toma como parámetro de entrada un gazetteer, un string que representa la etiqueta y la oración a analizar. Por cada palabra en el gazetteer la función busca la ocurrencia de ésta en la oración y la reemplaza por la etiqueta deseada. En el siguiente pseudocódigo podemos ver como está implementada esta función

```
1
2 \\entradas: gazetteer, etiqueta e, oracion
3
4 por cada palabra p del gazetter
5   Crear expresion regular para p
6   si la p ocurre al principio del texto
7     verificar inicio
8   si no reemplazar palabras que cumplan con la expresion regular por la
       etiqueta e
```

Observemos que al utilizar esta función podemos crear fácilmente subetiquetadores específicos para cada etiqueta simplemente instanciando la función con cada gazetteer y etiqueta que queremos. De esta manera, el tagger finalmente queda resumido en el siguiente código:

```
1 \\entrada: oracion
2
3 por cada gazetter y su correspondiente etiqueta e
4   oracion = findTags(gazetteer, e , oracion)
```

Es importante aclarar que para que el funcionamiento de los subetiquetadores sea correcto, no permitimos que una palabra pueda estar contenida en más de un gazetteer.

4.1.7. Etiquetas especiales

Para completar el sistema de reemplazo, se tuvo en cuenta que ciertos tags pueden ser identificados mediante expresiones regulares y sin contar con algún gazetteer. Aquí se muestra una breve reseña sobre ellos.

Precio:
Como se mencionó en el capítulo de la estrucutra del aviso, el precio del anuncio sigue una estructura fija que podemos capturar con una expresion regular. Básicamente identificamos patrones de números que comienzan o terminan con el caracter "$". El número en cuestión puede estar acompañado de otros caracteres como "." y ",".

Lugar :
Nuevamente, dada la estructura del aviso, podemos distinguir claramente el lugar presente en el título. Se identifica al lugar siempre rodeado de paréntesis y ubicado luego del precio del artículo.

Dimensiones :
En numerosos avisos se dan detalles de las dimensiones del objeto. La mayoría de éstas fueron extraídas también utilizando expresiones regulares.

Tiempo:
Se detectaron patrones que expresan un período mediante combinaciones de números y palabras claves como "<num> years old" entre otros.

4.1.8. Extracción de información

La información presente en las oraciones fue extraída con AdTagger y almacenada en formato Json para su manipulación posterior. En este punto es necesario discutir una nueva funcionalidad que fue incorporada al etiquetador, necesaria para detallar información más precisa referida al objeto.

Como se puede observar, en el proceso de desarrollo se optó por filtrar y almacenar palabras etiquetadas como sustantivos para identificar el objeto del cual hace referencia el aviso. Éstas, si bien reflejan componentes del objeto, individualmente no alcanzan para describirlo en su totalidad. Veamos un ejemplo para aclarar este hecho. Supongamos que la oración es la siguiente:

"New Cookie Cutter."

Aquí podemos distinguir dos sustantivos que hablan acerca del objeto central : "cookie" y "cutter". Ahora bien, nuestro etiquetador evaluaría la oración y la trasformaría de la siguiente manera:

"[STATUS] [OBJECT] [OBJECT]."

Los datos entonces para este caso son:
estado : “New”
Objeto: “Cookie” ; “Cutter”

Si bien esta salida es correcta, es necesario tomar algunas medidas para interpretar correctamente la información: la oración no está hablando de dos objetos distintos, “Cookie Cutter” es la interpretación correcta. Por este motivo se le incorporó al etiquetador la función de agrupar etiquetas de objetos e identificar las correspondientes palabras en el texto para que la salida puede ser evaluada correctamente. De esta manera, la salida real que muestra el programa es la siguiente:

“[STATUS] [OBJECT].”
y los datos son:
estado : “New”
Objeto: “Cookie Cutter”

4.1.9. Banco de templates

Una vez desarrollado el sistema de etiquetado AdTagger nuestra siguiente tarea consistía en generar un banco de “plantillas” o “templates” para utilizar en el proceso de generación. Para nuestro sistema, un template será una oración cuyas entidades de dominio han sido reemplazadas por las etiquetas correspondientes a cada clasificación de las entidades. Por ejemplo, si consideramos la oración “This is an antique lamp.” y extraemos sus entidades, generamos el siguiente template:
“This is an [STYLE] [OBJECT].”

De esta manera se procedió a ejecutar el algoritmo AdTagger para cada oración del corpus y se generó una gran cantidad de templates que almacenamos para usar posteriormente en las siguientes etapas.

4.1.10. Evaluación

Una vez desarrollado el algoritmo para etiquetar los avisos, nos propusimos a evaluar su funcionamiento utilizando el método de precisión y cobertura. De las 9498 oraciones seleccionamos al azar 100 de ellas que utilizamos como muestra para esta etapa de evaluación. Estas oraciones representan a 20 avisos completos y corresponden a varios rubros distintos con el fin de obtener una análisis más detallado del funcionamiento del etiquetador en cada categoría. Se analizó manualmente cada una de las 100 oraciones elegidas etiquetando las entidades que podían evidenciarse en ellas. Posteriormente se procesó cada oración con el algoritmo AdTagger y se procedió a comparar los resultados. Para esta tarea consideramos que las etiquetas marcadas durante el proceso de inspección manual son las

etiquetas correctas que debe tener cada oración. Luego se dispuso a determinar la cantidad de etiquetas marcadas por el algoritmo que resultaron válidas para cada categoría.

De esta manera pudimos determinar que el algoritmo encuentra 337 etiquetas el conjunto total de oraciones mientras que la cantidad real de etiquetas es 336. Por otro lado la cantidad de etiquetas que AdTagger detectó de manera correcta es 302. Con estos datos podemos observar que la **precisión** del etiquetador es de **0,89** y el **recall** del mismo es de **0,89**.

En el siguiente cuadro mostramos la distribución de las etiquetas detectadas correctamente por el algoritmo según la categoría a la que pertenece y además mostramos cómo contribuye cada subetiquetador (en porcentaje) a la precisión total.

sub etiquetador	**correctos**	**contribución**
object	101	29 %
name	4	1 %
colour	7	2 %
size	4	1 %
status	36	10 %
contact	4	1 %
feature	11	3 %
time	7	2 %
style	18	6 %
price	46	13 %
location	35	10 %
payment	2	1 %
material	11	3 %
dimension	11	3 %
sex	1	0 %
nationality	4	1 %

Cuadro 4.5: Distribución de etiquetas detectadas según categoría y contribución de cada subetiquetador a la precisión total

Como puede verse en el cuadro, los sub-etiquetadores que determinan el objeto, precio y lugar en el aviso son los que más contribuyen a la precisión total del AdTagger.

4.1.11. Ejemplos

Finalmente vamos a mostrar algunos ejemplos de avisos etiquetados por el algoritmo. A continuación mostramos oraciones de tres avisos que han sido empleados por AdTagger para construir los respectivos templates de cada una.

Ejemplo 1
Original:

Flower paintings / prints vintage Mrs. Loudon - $60 (san jose south) :
I have vintage Mrs. Loudon flower prints.
Vintage with frame.
60$ each obo.
Call phone #.

Templates:

[$OBJECT$] / [$OBJECT$] [$STYLE$] [$NAME$] - [$PRICE$] [$LOCATION$] :
i have [$STYLE$] [$NAME$] [$OBJECT$]
[$STYLE$] with [$OBJECT$].
[$PRICE$] each obo." [CONTACT] phone #.

Ejemplo 2
Original:

pair of chair - $350 (pacific heights) :
Antique Chair, Excellent condition.

Templates:

pair of [$OBJECT$] - [$PRICE$] [$LOCATION$] :
[$STYLE$] [$OBJECT$], [$STATUS$] condition.

Ejemplo 3
Original:

Antique etching by Edwin Fisher - $75 (morgan hill) :
This beautiful castle etching was valued at $130.
I'd accept $75 or best offer.
It was done by Edwin Fisher late 1800's early 1900's

Templates:

[$STYLE$] [$OBJECT$] by [$NAME$] - [$PRICE$] [$LOCATION$] :
this [$STATUS$] [$OBJECT$] was valued at [$PRICE$].
i'd accept [$PRICE$] or best offer.
it was done by [$NAME$] late [$DIMENSION$]s early [$TIME$]'s

Podemos ver que en la cuarta oración de los templates generados por el ejemplo 3, el algoritmo etiquetó incorrectamente la entidad "1800" como $DIMENSION$. ■

Clustering

Una vez desarrollado el AdTagger, nuestra siguiente tarea es agrupar oraciones con el fin de identificar distintas semánticas' que podemos encontrar presentes en los textos. El objetivo es poder identificar distintos temas o conceptos de los que se está hablando en las oraciones. Para tal fin fue necesario encontrar alguna representación semántica de las oraciones que fuera válida para poder compararlas y crear los grupos. Se procedió a evaluar dos alternativas con el fin de seleccionar la mejor de ellas.

5.1. Boxer

La primera alternativa evaluada fue usar Boxer para capturar la semántica del corpus. Esta es una herramienta creada por Johan Bos ([1] y [2]) que genera representaciones semánticas. Utiliza como inputs derivaciones CCG y produce como salida DRSs (véase [14]). Para su funcionamiento consta de un POS tagger y un banco de representaciones semánticas del CCGbank ([15]).

Cada DRS es una combinación de entidades generales del dominio (fecha, persona) y predicados que están relacionados mediante distintos elementos relacionales, típicamente funciones gramaticales (in,by). Utilizando dicha herramienta, se procedió a crear DRSs correspondientes a cada oración. Con esta información, se propuso extraer sólo los predicados identificados por Boxer. La suposición aquí es que los predicados capturan el significado conceptual de cada oracion. Para mostar un ejemplo de la semántica que puede extraerse de Boxer mediante predicados, veamos las siguientes oraciones:

- "Make offer for one or all."
  Algunos predicados extraídos del DRS de esta oracioń son:
  "make | all | offer | thing"
- "I know why they sell for but I have to get it gone I'll consider all decent offers, Thanks."
  Podemos extraer los siguientes predicados de su DRS: "thing | person | thing | person | offers | decent | consider | reason | know | have | go | get | know | sell"

Utilizando esta información, decidimos representar la oración mediante su correspondiente string de predicados.

5.2. Cadenas de tags

Como segunda alternativa, podemos observar que nuestro algoritmo AdTagger captura en efecto determinadas categorías semánticas de las oraciones. Utilizando este hecho, nuestra suposición es que podemos generar una representación bastante aproximada de la semántica real de la oración. Consideremos la siguiente oración:
"Antique Chair, Excellent condition."
Nuestro AdTagger reemplaza entonces tal oración por lo siguiente:
"[STYLE] [OBJECT], [STATUS] condition."

La disposición de las etiquetas en el ejemplo anterior muestra información importante para determinar el significado de la oración. Si sólo analizáramos el template sin conocer de dónde proviene, podríamos determinar que la oración está hablando de un objeto que posee un estilo determinado y que se encuentra en algún tipo de condición.

De esta forma, podemos representar la oración como la cadena de tags encontrados:
"STYLE , OBJECT , STATUS"

Notemos que si bien esta representación captura cierta semántica de la oración, depende fuertemente del funcionamiento del AdTagger y además de las presencias de tags en la oración. Si no es posible determinar etiquetas dentro de un aviso, esta representación no aporta ninguna información sobre el significado del mismo. Por otro lado, la semántica de esta cadena no incluye ninguna información sobre las demás palabras que no son etiquetadas que componen una oración.

5.3. Usando WEKA

Para ambas representaciones utilizamos WEKA para generar los clusters. Pero primero debemos tomar ciertos recaudos para que la herramienta pueda hacer clustering utilizando el algoritmo de K-means. En el caso de la representación creada por Boxer, primero transformamos la cadena de predicados en vectores utilizando la función StringToWordVector que nos provee WEKA. Dicha función produce atributos que representan la frecuencia de cada palabra en el string. Por defecto, cada palabra se convierte en un atributo cuyos valores pueden ser 1 o 0, reflejando respectivamente si la palabra está o no

en el string .

Para el segundo caso determinamos el número más grande de etiquetas presentes en una oración. Con este dato, transformamos cada string en una cadena de etiquetas de ese largo, completando los restantes tags con el signo "?" (que es interpretado por WEKA como atributo inexistente) y creamos tantos atributos como etiquetas.

A la hora de ejecutar K-means sobre este conjunto de oraciones, seleccionamos K suficientemente grande con el fin de sobre-generar clusters. Ésto facilita la verificación manual en caso de tener que mezclar clusters (en vez de dividirlos).

5.4. Análisis de resultados

Para analizar los clusters generados recurrimos a una inspección manual de cada cluster. Con este análisis pudimos determinar que los grupos creados en base a la representación basada en predicados de Boxer no eran tan similares como lo esperado.
Los distintos clusters resultaron difíciles de interpretar en términos de una semántica ingenua. A simple vista no pudo establecerse un patrón o concepto dentro de cada grupo. Se prodeció a repetir el experimento con k igual a 20, 40 y 50, obteniendo resultados similares a los antes mencionados. Concluímos que éste fenómeno observado se debía efectivamente a las formas de las oraciones que comprenden el cluster. Como mencionamos anteriormente, muchas veces las oraciones no poseen signos de puntuación o carecen de conectores para asociar palabras (típicamente "in", "by"), por lo que la estructura DRS de éstas es bastante errática.

Por otro lado los clusters creados en base a las cadenas de tags resultaron bastante más interesantes. Si bien cada grupo tenía oraciones muy ajenas al mismo, fue posible definir una semántica para cada grupo. Sólo bastó mezclar algunos clusters manualmente y filtrar aquellas oraciones que no correspondían a cada uno. Con esta tarea pudimos generar entonces varios clusters que finalmente usamos para continuar con nuestras tareas.

■

Recolección de estadísticas

En esta sección contamos cómo fue el procedimiento para obtener estadísticas de las oraciones y templates con el fin de usarlas en la etapa de generación y creación del modelo estadístico.

6.1. Identificando oraciones, templates y datos.

Como paso anterior a la extracción de estadísticas, se ordenó el material con el fin de facilitar las tareas de identificación de avisos, oraciones, datos y templates.

6.1.1. Oraciones

A cada oración del corpus se le agregó un identificador precedido del signo "#" el cual representa el número de cluster en el que se encuentra la cadena de etiquetas que representa a la oración. A su vez, al final de cada una se le anexó el carácter "@" seguido de un número único que representa a la oración unívocamente. De esta manera, para nuestros siguientes experimentos, una oración de un aviso tendrá la siguiente forma:

$\langle oracion \rangle ::= \langle string \rangle \ \#\langle clusterId \rangle \ \mathbf{@}\langle oracionId \rangle$

6.1.2. Templates

Para identificar los distintos templates se procedió de manera análoga a las oraciones, agregando a cada uno el número de oración al cual representa y el número de cluster al cual pertenece:

$\langle template \rangle ::= \langle string \rangle \ \#\langle clusterId \rangle \ \mathbf{@}\langle oracionId \rangle$

6.1.3. Datos

Como hemos mencionado cada oración tiene asociado un template y un conjunto de datos. Estos datos son almacenados en un arreglo identificado nuevamente por el número de oración a la cual pertenecen.

Oración : Flower paintings / prints vintage Mrs. Loudon - $60 (san jose south) :#51
Template: [OBJECT] / [OBJECT] [STYLE] [NAME] - [PRICE] [LOCATION] :#51
Datos : {OBJECT:[flower paintings,prints],LOCATION:[(san jose south)],NAME:[Mrs. Loudon],PRICE:[$60],STYLE:[vintage]} #51

6.1.4. Aviso

Por último, con los identificadores que agregamos para cada entidad, un aviso quedará conformado como una sucesión de oraciones comprendida entre los caracteres "{" y "}".

$\langle aviso \rangle ::= \{ \langle oracion \rangle + \}$

6.2. Estadísticas

Una vez determinados el formalismo necesario para identificar los componentes de nuestro sistema, procedimos a obtener un conjunto de estadísticos que utilizaremos fuertemente en la etapa de creación del modelo probabilístico que usamos para la generación. Es necesario primero resumir brevemente qué datos y materiales tenemos disponibles en esta etapa para aportar claridad al lector.

- Disponemos de un conjunto de clusters y la asignación de oraciones a cada uno de ellos.
- Tenemos un banco de templates organizados según cluster y según oración a la que corresponden.
- Contamos con la representación en forma de cadena de tags de cada template.

Con toda esta información disponible nos concentramos en obtener los siguientes estadísticos:

6.2.1. Distribución de frecuencia de Clusters por posición

Dada la posición de una oración en el aviso, calculamos con qué frecuencia el cluster al cual pertenece la oración cae en esa posición. Para ello utilizamos la siguiente ecuación: Sea C un cluster y P una posición, la frecuencia de C en la posición P será :

$$F_{C,P} = \frac{cantidad\ de\ oraciones\ del\ cluster\ C\ en\ la\ posicion\ P}{cantidad\ de\ oraciones\ en\ la\ posicion\ P}$$

6.2.2. Distribucion de frecuencia de templates por cluster

Para cada template encontramos la cadena de tags que lo representa y computamos la frecuencia de esta cadena en cada cluster. Sea t un template y CT la función que transforma el template en la cadena de tags correspondiente, la distribución de frecuencia de t en un cluster C será:

$$F_{C,t} = \frac{cantidad\ de\ ocurrencias\ de\ CT(t)\ en\ C}{cantidad\ de\ elementos\ en\ C}$$

6.2.3. Distribucion de Bigramas en Templates

Utilizando la cadena de tags que representa a un template, extraemos los bigramas que se encuentran en dicha cadena y calculamos la cantidad de ocurrencias de cada bigrama en los templates. Con estos datos calculamos la probabilidad de que un tag ocurra en un template dado que ocurrió otro antes. Para ello nos valimos del siguiente sustento teórico:

Dado un bigrama (C, D) donde $\{C,\ D\}$ pertenecen al conjunto de tags, la probabilidad de ocurrencia de D dado que ocurrió C es:

$$P(D|C) = \frac{cantidad\ de\ ocurrencias\ de\ (C,D)\ en\ templates}{\sum_{K \in tags} cantidad\ de\ ocurrencias\ de\ (C,K)\ en\ templates}$$

6.2.4. Distribución de Unigramas en Templates

Nuevamente utilizamos la cadena representante de cada template y calculamos la cantidad de ocurrencias de cada tag en los templates. Por último calculamos la probabilidad de ocurrencia de un unigrama C de la siguiente manera:

$$P(C \text{ ocurra en algun template}) = \frac{\text{cantidad de ocurrencias de } C \text{ en templates}}{\sum_{D \in tags} \text{cantidad de ocurrencias de } D \text{ en templates}}$$

■

Construcción del modelo de clasificación

Con el propósito de entrenar un modelo de clasificación, hicimos una división 70/30 de los datos para entrenamiento y testing. Representamos cada aviso como una secuencia finita de oraciones junto con los respectivos datos de cada una de ellas.

Dada una oración de entrenamiento junto con su template original y su conjunto de datos, primero seleccionamos aquellos templates de nuestro banco de templates que se corresponden con las mismas etiquetas presentes en los datos y que además tienen el mismo número de entidades para cada una. A este grupo de templates le extraemos una serie de características:

Distancia respecto del template original : Calculamos la distancia Levenshtein respecto del template sacado de la oración original.
Tamaño de template : cantidad de caracteres del template
Probabilidad de ocurrencia de tags en el template : evaluamos la probabilidad de que ocurra la secuencia de tags presentes en el template. Para ello nótese que si en un template ocurren los tags A, B, C y D en ese orden, la probabilidad de ocurrencia de esa secuencia de tags es $P(A) * P(B|A) * P(C|B) * P(D|C)$.
Probabilidad de ocurrencia de template dada posición : utilizando la cadena de tags que representa al template calculamos la probabilidad de ocurrencia de esa cadena dada la posición. Nótese que esta probabilidad es P(template dado cluster)* P(cluster dada posición).

Con esta información creamos nuestro modelo compuesto por una matriz 6648 x 4. Cada fila del modelo está conformado por un vector de cuatro dimensiones que representa a cada template, el cuál se compone de la siguiente manera: la primera coordenada contiene la distancia Levenshtein entre el template y el original. La segunda contiene el tamaño del template determinado en cantidad de caracteres que posee. La tercera coordenada contiene la probabilidad de ocurrencia del conjunto de tags que tiene el template y la cuarta coordenada corresponde a la probabilidad de ocurrencia del template dada la posición.

Para entender de manera más sencilla cómo está comprendida cada fila del modelo veamos el siguiente ejemplo: Supongamos que tenemos la oración Or que está ubicada en la posición pos de un aviso usado para entrenar. Sea $temp_1$ el template que obtenemos de Or extrayendo las entidades y sea $temp_2$ algun template que posee las mismas etiquetas presentes en $temp_1$ y además tiene la misma cantidad para cada una de ellas. El vector representante de $temp_2$ será entonces: $v(temp_2) = (LevenshteinDistance(temp_1, temp_2), size(temp_2), P(o$

El propósito de este modelo es analizar cómo contribuye cada componente de los vectores al modelo completo. Para este fin utilizaremos la herramienta de SVM disponible en WEKA, la cuál nos dará información del peso estimado de cada atributo del vector. En WEKA es posible determinar estos pesos analizando el polinomio característico que nos muestra como resultado en la salida del programa SVM.
Como nuestro modelo utiliza vectores tetra-dimensionales, este polinomio tendrá la la forma de $A_1W_1 + A_2W_2 + A_3W_3 + A_4W_4$, donde cada A_i representa un atributo del vector y W_i el peso asociado al mismo. Esta información nos será útil en la siguiente sección para determinar qué template usamos para generar.

■

AdGen

En este capítulo abordaremos el algoritmo que se desarrollo con el fin de generar textos referidos al dominio presentado a lo largo del trabajo. AdGen es un módulo escrito en su totalidad en Php. Recibe un aviso de entrada y genera a partir de éste otro aviso semánticamente equivalente. Está compuesto de varias funciones y módulos que detallaremos en las subsecciones siguientes.

8.1. Desarrollo

AdGen es una aplicación que se desarrolló utilizando técnicas descritas por Howald, Kondadadi y Schilder en [12]. Dado que el algoritmo puede ser extenso de analizar, presentamos a continuación una síntesis de su funcionamiento, dejando para las secciones siguientes los detalles de los módulos y otras funciones que acompañan a AdGen. Brevemente, nuestro algoritmo procede de la siguiente manera:

1. Recibe como parámetro de entrada el aviso que servirá de fuente para generar texto junto con el rubro al cual pertenece el aviso.
2. Divide al aviso en oraciones.
3. A cada oración se le aplica el AdTagger con el fin de identificar en la misma las entidades del dominio que contiene y a su vez genera un template para cada una de ellas.
4. El siguiente paso es recolectar para cada oración todos aquellos templates (del banco de templates que creamos en las etapas previas) que matchean en cantidad y tipo de entidades respecto a la información extraída de cada una.
5. Para cada uno de los templates se computa el conjunto de métricas y estadísticas que mostramos en el capítulo anterior. Para ello, cada template es reducido a la cadena de tags que contiene y se genera un vector representante de cada uno.
6. Utilizando el modelo, rankeamos los templates usando los pesos que obtuvimos para cada atributo y seleccionamos el mejor para cada oración.

7. Por último rellenamos los templates seleccionados con la información correspondiente a cada oración a la que representan. Este conjunto de oraciones creadas será entonces el texto generado en base al aviso original.

El algoritmo elaborado consta de varios módulos que operan para transformar y evaluar cada una de las oraciones que conforman el aviso. En las subsecciones siguientes detallaremos algunos de los módulos más importates que forman parte de AdGen.

8.1.1. TemplateMatch

Una de los algoritmos desarrollados para Adgen es TemplateMatch cuya finalidad es la de verificar si un template se corresponde en tipo y cantidad de etiquetas a un conjunto de datos. Éste puede ser resumido de la siguiente manera:
Como entradas el algoritmo recibe el conjunto de datos y el template a evaluar. Lo primero que hace es detectar el numero y tipo de etiquetas presente en los datos. Posteriormente se divide la verificación en dos etapas: en la primera se corrobora que los tipos de etiquetas del template sean exactamente aquellos que están en los datos. Para ello transforma el template en una cadena de etiquetas, similarmente al procedimieto efectuado a la hora del clustering. Con esta representación y usando funciones de Php es posible determinar si el template posee sólo los tags deseados, eliminando todas las ocurrencias de las etiquetas extraídas de los datos y verificando si la cadena resultante es vacia o no.

En la segunda etapa se comprueba si el número de entidades por cada etiqueta es igual en el template y en los datos. Para esto el algoritmo cuenta las ocurrencias de cada uno de los tags originales en la cadena de etiquetas. Si alguno de ellos es distinto,el template no cumple con las características de los datos.

```
1  entradas: template, datos
2  etiquetas = etiquetas de los datos
3  tempSring = tags del template
4
5  por cada etiqueta e
6      eliminar ocurrencia de e en template
7  si tempsString es vacio  -> continuar
8  sino ->  retornar false
9
10 por cada etiqueta e
11     c = ocurrencias de e en datos
12     d = ocurrencias de e en template
13     si c==d -> continuar
14     sino -> retornar false
15
16 retornar true
```

8.1.2. GetSentenceVectors

Como mencionamos en el cuarto ítem del procedimiento que ejecuta AdGen, para cada oración que conforma el aviso se seleccionan todos los templates,del banco que creamos, que matcheen en cantidad y tipo de entidades con los datos que se extraen de cada una, y por cada uno de de estos templates, se calcula un vector representante que contiene información de cuatro estadísticos mencionados en 6.2. Todas estas tareas son efectuadas por la función GetSentencesVectors cuyo funcionamiento se resume de la siguiente manera.
La función recibe como parámetros de entrada un template que corresponde a una oración del aviso junto con la posición que ocupa dentro del mismo. A su vez recibe el conjunto de datos que posee la oración. Primero filtra todos los templates que matchean con los datos de entrada. Luego, para cada uno de esos templates seleccionados calcula:

- distancia Levenshtein respecto del template original
- tamaño del template seleccionado
- probabilidad de ocurrencia de tags en el template seleccionado: para ello transforma el template en una cadena de tags y extrae los unigramas y bigramas de la cadena. Posteriormente calcula la probabilidad de ocurrencia de esa cadena de tags en el template.
- probabilidad de ocurrencia del template seleccionado dada la posición.

Por último, a cada atributo mencionado arriba lo multiplica por los respectivos pesos obtenidos del modelo y retorna el conjunto total de vectores.

A continuación mostramos en pseucódigo el funcionamiento del algoritmo.

```
1  \\entradas: template, datos, pos
2  \\ salida: vectores de templates que matchean con template
3
4  matches = templates que matchean con datos
5  vectores = arreglo vacio
6  por cada match en matches
7      vector[0] = distancia Levenshtein entre template y match * peso de
           atributo 0
8      vector[1] = tamanio de match * peso de atributo 1
9      tags = cadena de tags de match
10     vector[2] = probabilidad de tags * peso de atributo 2
11     vector[3] = probabilidad de match dada pos * peso de atributo 3
12
13     push(vectores, vector)
14
15 return vectores
```

8.1.3. Funciones auxiliares para el cómputo de métricas y estadísticas

Para calcular los estadísticos que se generan para cada template, elaboramos funciones auxiliares que ayudan a computar el valor de cada uno. Entre las funciones más importantes podemos destacar las siguientes:
GetAllBiGrams: Ésta es la encargada de detectar todos los bigramas presentes en el template.
GetFrecuencyCUIDByPos: Función encargada de determinar cuál es la distribución del cluster del template dada la posición en el aviso.
GetFrecuencyTempByCUID: Su tarea consiste en determinar la distribución del template según el cluster al que pertenece.

8.1.4. Selección del mejor template

Como hemos mencionado para cada oración que contiene el aviso, seleccionamos los templates que matchean y a cada uno de estos les calculamos su vector representante. El siguiente paso es determinar qué template es el mejor o más adecuado para usar en la etapa de generación. Para seleccionar cuál es el vector más adecuado determinamos cuál es el vector que minimiza las primera dimensión y maximiza las restantes tres de entre todos los vectores disponibles.

8.1.5. TemplateFill

Esta función tiene por objetivo rellenar un template dado con un conjunto de datos establecido. Para ello primero inspecciona el template de entrada con el fin de verificar si el número y tipo de tags presentes en él son los mismos que los que contiene la información. En caso de no poder establecer estas igualdades, la función devuelve el template sin rellenar.
El mecanismo básico de la función recae en recortar uno a uno los tags que se quieren reemplazar del template para sustituirlo por el dato corresponiente. En el siguiente pseudo código se puede ver un poco más en detalle el funcionamiento de TemplateFill

```
1 \\entradas: template, datos
2 \\ salida: oracion formada con template y datos
3 etiquetas = etiquetas de los datos
4 oracion = template
5 por cada etiqueta en etiquetas
6     entidades = entidades de la etiqueta en datos
7     por cada entidad en entidades
8       pos = primera ocurrencia de etiqueta en oracion
```

```
9         oracion = oracion[0,pos] + entidad + oracion[size(etiqueta),size(
              oracion)]
10
11  return oracion
```

8.1.6. Algoritmo

Por último mostramos en pseudocódigo cómo se compone AdGen utilizando los distintos módulos mencionados.

```
1  \\Entradas : aviso, rubro
2  oraciones = oraciones del aviso
3  avisoGen = arreglo vacio
4  por cada oracion en oraciones
5    arreglo = AdTagger(oracion,rubro)
6    templateOracion = arreglo[0], datos = arreglo[1], pos = posicion oracion
         en aviso
7    matriz = GetSentencesVectors(templateOracion, datos, pos)
8    normas = arreglo vacio
9
10   por cada vector en matriz
11     push(normas,norma(vector))
12
13   max = norma mas grande del conjunto normas
14   template = template con norma max en matriz
15
16   push(avisoGen, TemplateFill(template, datos))
17
18 return avisoGen
```

8.1.7. Ejemplos de avisos generados

A continuación mostramos algunos ejemplos de avisos que han sido generados utilizando AdGen. Para generarlos hemos dado como datos de entrada avisos de distintas categorías y las entidades presentes en ellos han servido para rellenar los mejores templates que el sistema determinó y crear estos nuevos avisos.

Original / generado:

1. Vintage with frame. 60$ each obo. Call please.
   / Vintage millifiore frame - 60$: If you would be interested in either or both opportunities, please call me for additional information.

2. Antique etching by Edwin Fisher - $75 (morgan hill) : This beautiful castle etching was valued at $130. I'd accept $75 or best offer.

/ Antique etching by Edwin Fisher - $75 (Morgan Hill) : you can get one for around $130 on ebay and you get yourself a beautiful castle etching that last forever. Selling dont need it no more any offer will be taken into consideration thanks!! $75 obo

3. easy washing machine old antique - $850 (gilroy) : easy model . M.washing machine ..works good would be beautiful polished,cooper brass nickel plated inside .
   / Antique washing machine - $850 (Gilroy) : Brand new beautiful good includes instructions, M. washing machine, polished brass polishedcooper included and nickel.

4. Antique Oak Desk with Chair - $50 (redwood city) : Solid oak antique desk.Dimensions are 49" long 31" wide.
   / Antique primitive hand craved oak desk trough chair - $50 (Redwood City) : I have a solid oak antique desk. I believe one is 49" and the other is 18 or 31" .

5. 30 metal cookie cutters - $30 (santa cruz) : 30 cutters. $30 firm.
   / Metal cookie cutters- $30 (Santa Cruz) : We are selling our cutters for $30 each, and will offer a discount if you buy more than one.

6. Antique Chairs - $65 (willow glen / cambrian) :Two beautiful antique chairs, one in beautiful condition and one with a broken wheel at the base of the leg.
   / Antique indutrial hospital side chairs - $65 (Willow glen / Cambrian) : Antique seagram's chairs with beautiful wheel in leg and in broken condition.

7. French Baby Desk - $50 (petaluma) : French baby desk for sale.
   / museum desk. - $50 (petaluma) : has been ridden french and could easily go in that direction as well, has some basic desk reining training.

Como podemos observar en los ejemplos generados, nuestro sistema no es capaz de identificar el número gramatical en los avisos. Es por ello que en ejemplos como 4 se puede ver que el anuncio comienza hablando de un sólo producto pero en las siguientes oraciones se deduce que se está hablando de varios. Además podemos observar en el ejemplo 3 cómo la ausencia de signos de puntuacíon dificulta la interpretación del aviso. También es posible ver que en general los avisos generados no conservan toda la semántica del aviso original, ésto es probablemente debido a que las cadenas de tags no son suficientes para capturar el significado de las oraciones.

■

Conclusiones

9.1. Sobre el corpus en general

Los avisos clasificados que extrajimos de Internet para realizar el presente trabajo presentaba errores y detalles que fueron determinantes para los resultados obtenidos del etiquetador no supervisado y los avisos generados.

Abreviaciones
Uno de estos detalles es la existencia de abreviaciones de palabras inventadas o no estándares. En muchas oportunidades las oraciones de los avisos tenian ocurrencias de abreviaciones de sustantivos que ni TreeTagger ni Wordnet reconocían. Esto contribuyó a la pérdida de entidades y datos de los avisos.
En otros casos, las abreviaciones fueron aceptadas erróneamente por TreeTagger como sustantivos o adjetivos , lo que produjo que ciertos templates generados no sean semánticamente correctos.

Signos de puntuación
Otra de las problemáticas del corpus tiene que ver con la presencia o ausencia de signos de puntuación. Es posible ver en algunos avisos que se omiten signos de puntuación importantes a nivel gramatical como lo son los signos de fin de oración y comas. Esto repercutió de manera considerable en los análisis efectuados por las herramientas como TreeTagger y, por sobre todo, de Boxer.

Palabras mal escritas
Como mencionamos anteriormente los avisos del corpus tienen ocurrencias de palabras del idioma Inglés mal escritas. Se utilizó un corrector ortográfico para intentar solucionar ese problema pero no fue suficiente para corregirlas a todas. De esta manera quedaron palabras importantes para la semántica del aviso que al estar mal escritas no pudieron ser clasificadas por el etiquedator.

9.2. Sobre AdTagger

El desarrollo de este algoritmo aportó ideas interesantes para construir taggers no supervisados específicos del dominio de manera semiautomática. El método de analizar los avisos utilizando TreeTagger para filtrar sustantivos y adjetivos para luego clasificarlos utilizando Wordnet es un buena aproximación para elaborar un tagger no supervisado y además nos permite analizar hasta qué punto se puede automatizar este proceso de selección e identificación de entidades en un corpus.
Otra ventaja del método propuesto para elaborar el tagger es que puede ser fácilmente adaptado a otros dominios. Notemos que con la metodología utilizada la mayor parte de las entidades reconocidas por el tagger fueron determinadas automáticamente. Ésto ofrece la ventaja de no necesitar expertos en ligüística para resolver las tareas de etiquetado o comprender la sintaxis (incluyendo vicios del lenguaje) de las personas que escriben clasificados.
Por otra parte, el algoritmo AdTagger puede configurarse fácilmente para incorporar otras etiquetas: en el trabajo espećifico que hemos realizado sólo nos centramos en un subconjuto de todas las etiquetas que podemos encontrar en los avisos. Esto se debe a que nuestros intereses recaían específicamente en el método por el cual reconocemos etiquetas automáticamente y no tanto en realizar un análisis completo de entidades en el dominio. De esta forma dejamos de lado otras etiquetas que pueden ser estudiadas y reconocidas, y ademas pueden agregarse sencillamente en el etiquetador. Como hemos mencionado en el capítulo del desarrollo del algoritmo, este se basa en una serie de funciones que cumplen el rol de subetiquetadores y por lo tanto, definiendo una nueva función asociada con un nuevo gazetteer es posible generar otro subetiquetador que puede ser usado por el AdTagger.
El proceso que desarrollamos para crear este etiquetador incorporó también errores generalmente causados por la automatización de la creación de los gazetteers. Una desventaja importante es la fuerte dependencia que existe con las herramientas usadas para crear los archivos fuentes de cada subetiquetador. Si una palabra no es reconocida por TreeTagger o Wordnet, esta se descarta, lo que puede llegar a suponer un error. Una posible solución a este problema radica en utilizar técnicas de Collocation Detection para intentar encontrar las entidades dentro de los textos.

Otro aspecto importante que queremos recarmar sobre el desarrollo de este algoritmo es la combinación de dos grandes metodologías usadas para elaborar etiquetadores basados en corpus: aproximaciones lingüísticas y aproximaciones de aprendizaje automático. Notemos que muchas entidades que logra reconocer el tagger son extraídas y filtradas utilizando TreeTagger el cual utiliza modelos probabilísticos para determinar la categoría de vocablos, hecho reconocido como una aproximación al reconocimiento de etiquetas utilizando aprendizaje automático. Sin embargo como hemos mencionado repetidas veces utilizamos expresiones regulares para reconocer otras etiquetas como precio, dimensiones o lugar las cuales constituyen reglas puramente sintácticas. Éstas no fueron reconocidas mediante algún algoritmo o método automático, si no que fue necesario un análisis puramente manual.

Una limitación importante de este algoritmo tiene que ver con la identificación de número de sustantivos, pronombres personales, pronombres posesivos , verbos y tiempos verbales. Esta información es muy útil para entender y analizar un texto. En nuestro proyecto este tipo de identificación fue omitido voluntariamente puesto que la fuente que manejamos, en este caso avisos clasificados, no posee una complejidad lingüística elevada y con la identificación de etiquetas que mostramos consideramos que es suficiente. Sin embargo, si se desea utilizar el etiquetador en otros dominios lingüísticamente más complejos este tipo de información debe ser tenida en cuenta.

9.2.1. Análisis de etiquetas según rubro

Uno de los objetivos del trabajo en general era aprender de qué manera se estructura un aviso y cuáles son sus componentes principales. Por medio de AdTagger podemos analizar cómo influyen las distintas categorías de etiquetas en la composición de un aviso clasificado según el rubro en el cual está comprendido y a su vez, identificar que entidades son más utilizadas para describir cada tipo de aviso. En el siguiente cuadro mostramos de qué manera contribuye cada etiqueta (en porcentaje) a la constitución del aviso.

Etiqueta	atiguedades	muebles	prod. belleza	colección	prod. granja	hogar	joyas
OBJECT	38.05 %	42.15 %	52.09 %	53.04 %	52.88 %	42.51 %	45.23 %
NAME	0.75 %	0.46 %	0.12 %	0.25 %	0.21 %	0.44 %	0.64 %
COLOUR	1.37 %	2.66 %	0.73 %	1.52 %	1.48 %	2.4 %	7.15 %
SIZE	2.59 %	2.51 %	1.4 %	2.02 %	2.47 %	2.59 %	1.72 %
STATUS	8.85 %	11.84 %	11.28 %	11.76 %	8.9 %	11.86 %	10.21 %
CONTACT	1.67 %	0.76 %	1.62 %	1.35 %	1.82 %	1.76 %	0.73 %
FEATURE	5.87 %	6.35 %	5.75 %	6.84 %	5.87 %	5.66 %	5.34 %
TIME	3.76 %	1.74 %	0.58 %	1 %	2 %	1.38 %	1.98 %
STYLE	6.56 %	2 %	0.63 %	0.37 %	0.68 %	1.4 %	1.63 %
PRICE	8.53 %	10.82 %	11.64 %	11.09 %	9.4 %	11.46 %	9.78 %
LOCATION	8.14 %	7.64 %	8.43 %	7.55 %	7.01 %	8.54 %	7.32 %
PAYMENT	0.53 %	0.41 %	0.56 %	0.39 %	0.41 %	0.53 %	0.86 %
MATERIAL	4.99 %	4.51 %	1.76 %	1.16 %	1.78 %	4.24 %	4.13 %
DIMENSION	5.56 %	5.23 %	3.24 %	1.48 %	3.94 %	4.76 %	2.67 %
SEX	0 %	0 %	0 %	0 %	0 %	0 %	0 %
NATIONALITY	2.72 %	0.87 %	0.1 %	0.1 %	0.96 %	0.4 %	0.51 %

Cuadro 9.6: Distribución de etiquetas según rubro.

Con este cuadro podemos ver que, en general, las etiquetas de OBJECT,STATUS,PRICE y LOCATION son las que más relevancia tienen dentro de un aviso. Esto se corresponde claramente con la información más básica e importante que puede proveer un anunciante: hay un objeto OBJECT para comercializar, cuyo precio es PRICE, que se encuentra en STATUS condición y está en LOCATION. Por lo general esta información está contenida en el título del aviso y no es casualidad ya que es lo primero que el lector nota y en base a esto intentará filtrar avisos según si cumplen o no con sus necesidades o posibilidades.

Por otro lado, es interesante ver como el rubro determina qué información es importante transmitir y cuál no: si se intenta vender joyas es más probable que se detalle de qué color es y de qué material está hecho, mientras que si el objetivo es vender algún bien mueble el anunciante dará también importancia al material del cuál esta constituido pero en este caso es más importante mencionar las dimensiones del objeto en lugar del color. De igual manera se puede observar que el estilo y el origen (tiempo) son características más importantes si se pretende publicitar una antigüedad.

9.3. AdGen

En esta sección listaremos debilidades y fortalezas del generador

- El generador no logra generar concordancia de número y género correctamente. Ésto se debe principalmente a que no identificamos aún con el AdTagger los pronombres personales y posesivos que son los que nos dan la idea de quién es el que realiza la acción de la oración. Por otro lado tampoco puede reconocer la cantidad de objetos de los que habla el anuncio.

- Como es un módulo que opera dependiendo fuertemente del etiquetador, cualquier error que éste pudiera cometer afecta a la generación de avisos. Un ejemplo de ésto se da cuando el tagger no reconoce ciertas entidades en el aviso. El generador busca siempre los templates que matchean en tipo y cantidad de etiquetas, por lo que si una éstas se omite, los templates selecionados no serán los adecuados.

- Otra debilidad del algoritmo tiene que ver con las posiciones de las etiquetas en el aviso y en los templates. El algoritmo que busca los templates que matchean (GetAllMatches) no tiene en cuenta la posición en la que están los tags dentro de las oraciones, sólo comprueba cantidad y tipos de tags por lo que al momento de generar podemos encontrar entidades que al cambiar de posición pierden sentido dentro de la oración.

- Por otra parte podemos destacar que el generador puede ser fácimente adaptado a otros dominios. Los procedimientos efectuados para elaborar AdGen no son dependientes del corpus a excepción del etiquetado. Utilizando algún otro etiquetador específico de otro dominio se pueden efectuar los demás procedimientos sin ningún inconveniente.

- También podemos ver que es fácil agregar nuevos estadísticos al modelo para incorporar nuevas opciones para la selección de los mejores templates. Los cálculos que efectuamos para generar el modelo son algunos de los recomendados en [12].

Agregar nuevos estadísticos sólo supone agregar otras dimensiones a los vectores representantes de los templates.

- En la etapa de clustering utilizamos las cadenas de tags como representantes de la semántica de las oraciones que conforman los avisos. Sin embargo ésto no es suficiente para capturar el significado de las oraciones dado que perdemos información semántica que está presente en el resto de las palabras que ocurren en ella. Una posible solución a este problema es utilizar otro analizador semántico para intentar capturar el significado de los avisos.

- Como pudimos ver en los ejemplos de AdGen, las categorías semánticas de los textos generados no se corresponden completamente con los textos fuentes. Creemos que este fenómeno es consecuencia de que el corpus tiene avisos muy diferentes entre sí en donde establecer relaciones entre ellos es difícil debido a la separación semántica que existe.

9.4. Trabajos Futuros

9.4.1. Sobre AdTagger

Uno de las tareas a realiza es ampliar el conjunto de etiquetas a detectar: agregar otros subetiquetadores que intenten detectar los pronombres personales y posesivos que ocurran en los avisos. De esta manera se podrá distinguir el número y género de personas que se mencionan en el aviso. A su vez incorporar reglas de plurales en sustantivos para poder ampliar el conjunto de objetos a identificar.
Este algoritmo puede paralelizarse por lo que se puede disponer la ejecución de cada subetiquetador en un hilo de procesamiento distinto.
Por otra parte la identificación de verbos dentro del texto fue omitido de manera intencional de nuestro sistema. Sin embargo esta identificación puede ser incorporada con el fin de realizar un análisis más exhaustivo del texto fuente que puede ser útil en la generación.
Un ejercicio interesante para realizar es aplicar la metodología de extracción de entidades a otros dominios o géneros con el fin de evaluar la precisión de este proceso semiautomático.

9.4.2. Sobre AdGen

Dado que las cadenas de tags no son suficientes para capturar el significado de los textos, proponemos utilizar otros analizadores semánticos con el fin de encontrar estructuras más expresivas o ricas que permitan contener más información sobre la semántica de los avisos.

Dado que el cómputo de templates que matchean con una oración es costoso y depende del tamaño de banco de templates del que se dispone, es posible crear hilos de ejecución por cada oración del aviso. También es posible agregar memoria cache para almacenar los templates que se corresponden con un conjunto de templates (es decir, con los templates que poseen los mismos tipos y cantidad de etiquetas) para ahorrar tiempo computacional.

Utilizando otro conjunto de etiquetas podremos generar avisos que respeten reglas gramaticales como número y género. Por otra parte mediante la identificación de los verbos en el texto mediante TreeTagger es posible incorporar a AdGen la opción de generar avisos que respeten los tiempos verbales y la sintaxis de los verbos según el sujeto. Además, contando con esta funcionalidad y la mencionada en el item anterior podemos incluir la opción de generar texto en singular o plural y además dirigido para una persona específica.

Como otra tarea futura proponemos ampliar el corpus significativamente con el fin de agregar más precisión a las categorías semánticas que buscamos mediante clustering. Creemos que este hecho logragrá una mejor aproximación entre la semántica del texto usado como fuente y el texto generado.

En la seccíon 4.1.8 mencionamos la dificultad de extraer entidades compuestas de varias palabras como “Cookie Cutter”. Este tipo de reconocimiento puede realizarse mediante una técnica conocida en Procesamiento de Lenguaje Natural denominada Collocation Detection.

También es posible incorporar a la etapa de selección de mejor template nuevas funcionalidades para que la elección no sólo tenga en cuenta la norma del vector representante, sino que que permitan penalizar los templates según el tamaño o la variación de distribución de etiquetas respecto al template original

Queremos destacar además que utilizando AdTagger para reconocer las entidades de los avisos es posible modificar el generador de avisos para lograr un anuncio parafraseado del original utlizando la misma metodología de clustering para detectar templates semánticamente similares.

Por último un actividad que dejamos pendiente como trabajo futuro es la evaluación del generador. Esta tarea no fue realizada debido a la magnitud del trabajo realizado sólo para crearlo.

■

Bibliografía

[1] James R. Curran,Stephen Clark and Johan Bos, *Linguistically Motivated Large-Scale NLP with C&C and Boxer.* Proceedings of the ACL 2007 Demonstrations Session (ACL-07 demo), pp.33-36. 2007

[2] Basile, V. and J. Bos, *Towards generating text from discourse representation structures.* Proceedings of the 13th European Workshop on Natural Language Generation (ENLG), pp. 145-150. 2011

[3] Helmut Schmid , *Improvements in Part-of-Speech Tagging with an Application to German.* Proceedings of the ACL SIGDAT-Workshop. Dublin, Ireland. 1995

[4] E. Reiter and R. Dale , *Building applied natural language generation systems.* Journal of Natural Language Engineering, 3(1):57-87, 1997.

[5] E. Reiter, S. Sripada J. Hunter and . Yu y I. Davy, *Choosing words in computer-Artificial generated weather forecasts.* Intelligence, 167(1-2): 137-169, 2005

[6] McKeown, Kathleen,*Text Generation.* Cambridge University Press,Cambridge.1985

[7] Goldman, Neil M , *Conceptual generation.* In Roger C. Schank and Christopher K. Riesbeck, editors, Conceptual Information Processing. American Elsevier, New York.1975

[8] Ian H. Witten and Eibe Frank, *"Data Mining: Practical Machine Learning Tools and Techniques"*, Second Edition 2005.

[9] Manning and H. Schütze, *Foundations of statistical natural language processing* - 1999

[10] Levenshtein, V., *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady 10, 707-710. 1966

[11] Anja Belz., *Probabilistic generation of weather forecast texts.* In Proceedings of Human Language Technologies 2007: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT' 07), pages 164-171. 2007

[12] Blake Howald,Ravi Kondadadi and Frank Schilder, *Domain adaptable semantic clustering in statistical NLG.*. In Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013), pages 143-154. Association for Computational Linguistics, March. 2013

[13] Joachims, T., *Learning to Classify Text Using Support Vector Machines.* Kluwer. 2002

[14] Kamp, H. and U. Reyle, *From Discourse to Logic; An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT.* Kluwer, Dordrecht. 1993

[15] Hockenmaier, J. and M. Steedman, *CCGB ANK : User's manual.* In Department of Computer and Information Science Technical Report MS-CIS-05-09. University of Pennsylvania, Philadelphia, PA.2005

[16] Manual PHP , `www.php.net/manual/es/index.php`. Marzo 2015

[17] The Penn Treebank Project , `www.cis.upenn.edu/~treebank/`. Marzo 2015

[18] Etiquetas del Penn Treebank Project , `www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`. Marzo 2015

[19] Davey, Anthony C., *Discourse Production.* Edinburgh University Press,Edinburgh, Scotland.1979