Detección de similitud en textos cortos considerando traslape, orden y relación semántica de palabras

Por

Miguel Ángel Álvarez Carmona

Tesis sometida como requisito parcial para obtener el grado de

MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE CIENCIAS COMPUTACIONALES

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla
2014

Supervisada por:

Dr. Jesús Antonio González Bernal
Investigador titular del INAOE

Dr. Manuel Montes y Gómez
Investigador titular del INAOE

A mi madre
una vez más

Resumen

En los últimos años se han creado grandes conjuntos de documentos digitales que pueden ser fácilmente consultados. Estas colecciones contienen obras que abarcan una gran variedad de temas con una enorme diversidad de enfoques. Al incrementarse de manera tan abrupta la cantidad de información ha surgido la necesidad de generar herramientas que, de forma automática, nos ayuden a administrar dicha información.

Existen herramientas para poder manejar el gran volumen de información disponible. Estas herramientas pueden ser: clasificadores temáticos de texto, sistemas de detección de plagio, detección de reputación en redes sociales, entre otros. Estos sistemas tienen algo muy importante en común; necesitan conocer el grado de similitud textual entre un par de textos.

El cálculo de similitud textual es fundamental para una amplia gama de tareas. Por este motivo, recientemente, la similitud textual ha recibido mucha atención por parte de la comunidad científica. Un problema particular de la detección de similitud textual, es la detección de similitud en textos cortos, una tarea un tanto más complicada que la original por la poca información con la que se cuenta para llevar a cabo la toma de decisiones.

En diversos trabajos se ha demostrado que los enfoques que dan buenos resultados para encontrar el grado de similitud en documentos largos, no consiguen ser suficiente

buenos o adecuados cuando se mueven al terreno de los textos cortos. Esto se debe a que los trabajos clásicos están basados en su mayoría en técnicas de frecuencia y probabilidad. Por este motivo, para detectar de forma automática la similitud en textos cortos es necesario desarrollar métodos más sofisticados.

En este trabajo de tesis se aborda la tarea de detectar el grado de similitud sobre textos cortos. Para esto, se proponen dos enfoques diferentes para llevar a cabo esta tarea.

La primer contribución de este trabajo es la propuesta de tres métodos no supervisados basados en algoritmos que, originalmente fueron creados para resolver la tarea de alineación de cadenas biológicas. Estos algoritmos están hechos para reconocer un par de secuencias y encontrar las principales semejanzas entre ellas. A estos algoritmos se les otorga información semántica para poder ser utilizados, ya no en secuencias biológicas, sino en textos.

Por el otro lado, la siguiente contribución de esta tesis es un método supervisado, basado en programación genética. Este algoritmo se alimenta de diversas medidas que por separado sirven para obtener información especifica de un par de textos. En total se utilizan 20 medidas basadas en información de traslape, orden y relación semántica de las palabras, de las cuales se intenta encontrar la mejor relación entre ellas para detectar el mejor grado de similitud posible. De esta forma se propone un método capaz de diseñar funciones de manera automática para detectar similitud textual.

En este trabajo se da evidencia de que los métodos propuestos dan resultados competitivos en tareas diferentes, aún cuando no se desarrollaron para una tarea específica. De esta forma se aportan métodos que funcionan adecuadamente para comparar textos cortos en general.

Abstract

In recent years, large sets of digital documents have been created and these can be easily viewed. These collections contain works with much variety of topics with a huge diversity of approaches. By so abruptly increased on the amount of information and knowledge has become necessary to build tools that automatically help us to manage this information.

There are very important available tools to manage the large volume of information. These tools include: thematic text classifiers, plagiarism detection systems, detection of reputation in social networks, among others. These systems have something very important in common; everyone needs to know the degree of textual similarity between a pair of texts.

The computing of textual similarity is a fundamental technique for a wide range of tasks in natural language processing. For this reason, recently the textual similarity has received much attention from the scientific community. A particular problem of textual similarity detection is the task of detection of similarity in short texts, a somewhat more complicated than the original by the little information with it has to carry out the decision-making.

Several studies have shown that the approaches are successful to find the degree of similarity in long documents are not suitable when applied to short texts. This is

because the classical works are based mostly on frequency and probability techniques. This is why to automatically detect the similarity of short texts is necessary to develop more sophisticated methods.

In this thesis the problem of detecting the degree of similarity on short texts is attacked. For this, two different approaches were proposed to carry out this task.

Firstly unsupervised methods based on algorithms that were originally created to solve the task of aligning biological chains are proposed.These algorithms are made to recognize a pair of sequences and find the main similarities between them. In this work these algorithms are given semantic information to be used, not in biological sequences, rather in texts.

On the other hand, a supervised method based on genetic programming is proposed.This algorithm feeds on several measures separately used to obtain specific information from a pair of texts. A total of 20 measures are used, of which we try to find the best compromise between them to identify the best possible degree of similarity. Thus, a method capable of designing functions automatically to detect textual similarity is proposed.

This thesis work gives evidence that the proposed method gives competitive results in different tasks, even when they were not developed for a specific task. Thus, methods that work well for comparing short texts generally are provided.

Agradecimientos

Mi agradecimiento, en primer lugar al pueblo mexicano y al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo otorgado a través de la beca no. 280653.

A mi madre, a la cual me faltarán tesis para poder agradecerle todo su apoyo incondicional a lo largo de toda una vida. ¡Gracias mamá!

A mis asesores, los Drs. Manuel Montes y Gómez y Jesús González Bernal mi más sincero agradecimiento por su apoyo constate, sus comentarios acertados y sus consejos que me acompañaron a lo largo de mis estudios de maestría en el INAOE.

A mis sinodales, Dr. Aurelio López López, Dr. Luis Villaseñor Pineda y Dr. Saúl Eduardo Pomares Hernández por sus observaciones y comentarios.

A Alan, Alfredo, Gibrán, Gustavo y Julio, que me he perdido de muchos momentos con ustedes por estar haciendo otras cosas (como una tesis) pero que de igual forma me entienden y sé que su amistad la tendré siempre.

A Silver, Cruz, Nius, Voro, mi tocayo, Malli, Neto, Valde, Laura, Luis, Efra, Mau,

los Ivanes, Erick, Polo, Blanca, Carlos, Jona, Jorge, incluso a Israel. Somos una gran generación, hubo grandes momentos entre clases, tareas, exámenes, retas y convivios. Me debo sentir muy afortunado por tener tanta gente a quien agradecerle tantas cosas.

A Zelma, nadie dijo que el camino sería fácil pero si a esta altura y pese a todo aún estamos caminando es porque a los planetas les gusta alinearse en cada viaje redondo y sin escalas de Mochis a Chachapa.

A quien se tome el tiempo de leer este trabajo de tesis.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En las últimas décadas se ha presentado un crecimiento exponencial en la cantidad de textos electrónicos disponibles en la Web; cuyo contenido, día a día, incrementa en cantidades dignas de pasajes bíblicos. Este auge de información digital ha contribuido a que, actualmente, podamos acceder desde cualquier dispositivo a casi cualquier tema en el mundo.

Aunque parezca increible, mientras usted está leyendo este párrafo, en Twitter se mandan poco más de 278.000 *tweets*, en Facebook se crean 350 GB de información, en Google se realizan dos millones de búsquedas, se crean 571 nuevas páginas web, se mandan 204 millones de correos electrónicos, en Instagram se suben 3.600 fotos y en WordPress se escriben 347 *posts*, ya que según el sitio Qmee[1], esto ocurre cada minuto en la web y la tendencia es que cada vez estas cifras serán mayores.

Debido a que la masa de información, crece de manera casi alarmante, cada vez es más difícil el manejo de la misma, lo cual sin mecanismos adecuados, lejos de hacer difícil el acceso y manejo de la información, lo haría imposible, lo que convertiría a este gran conjunto con el que contamos actualmente, en algo completamente inútil. Por esta razón,

[1]Visto el 15 de octubre de 2014 http://blog.qmee.com/qmee-online-in-60-seconds/

surge la necesidad de procesar automáticamente esta información para facilitar muchas tareas. Por tal motivo se ha recurrido a una de las ramas de la Inteligencia Artificial: el Procesamiento del Lenguaje Natural (PLN) (Manning & Schütze, 1999).

El objetivo de esta área es crear métodos, técnicas y herramientas computacionales que permitan realizar análisis de información escrita u oral y que faciliten la búsqueda y organización de dicha información (Holland *et al.*, 2013; Verspoor & Cohen, 2013).

Para facilitar el acceso y manejo de información, el PLN se ha encargado de crear sistemas para: agrupación de documentos (Chang *et al.*, 2014), sistemas de clasificación de texto (Li & Xu, 2014) así como sistemas para detectar plagio o semejanza entre documentos escritos (Ceglarek, 2013), entre otros. Todos estos sistemas antes mencionados tienen el mismo motor para su funcionamiento: la detección de similitud textual.

Una de las tareas más complicadas e importantes en el procesamiento automático del lenguaje humano es sin duda, la detección de similitud textual, tanto por la ambigüedad del lenguaje y su riqueza como por sus amplias aplicaciones y por la ayuda que aporta a la toma de decisiones en sistemas automáticos.

Se han llevado a cabo investigaciones acerca de la detección automática de similitud textual y se han logrado avances significativos en los útimos años. La mayoría de trabajos se enfocan en encontrar el parecido en documentos largos, donde se cuenta con una cantidad de información suficiente para tomar decisiones acerca de la cercanía o lejanía de los textos. Pero, existe una variante en la tarea, la cual consiste en detectar el grado de similitud en textos cortos como: tuits, mensajes de texto o títulos de noticias. Esta variante es una tarea un tanto más difícil que la original por la poca información con la que se cuenta para tomar decisiones (Liu *et al.*, 2007).

En este trabajo de tesis nos centraremos en el problema de detectar el grado de similitud textual sobre textos cortos.

1.1. Problemática actual

Detectar similitud textual sobre documentos largos es un problema parcialmente resuelto ya que representaciones simples como bolsa de palabras (Wu *et al.*, 2010) y otras que, posteriormente se han desarrollado en el contexto de la recuperación de información y clasificación de documentos, son bastante robustas y producen buenos resultados. Pero cuando se trata de evaluar textos cortos, estas representaciones no son ideales para lograr una buena efectividad, ya que, los esquemas de representación basados en frecuencia no funcionan correctamente cuando no hay suficiente información; y se tiene que recurrir a maneras más sofisticadas para hacer su comparación.

Existen diversas formas de abordar la tarea de detectar el grado de similitud en textos cortos. Algunos trabajos se basan en el traslape de conjuntos de palabras (Han & Baldwin, 2011), otras medidas consideran sustituciones o transformaciones semánticas (Šarić *et al.*, 2012) mientras que, otras consideran detectar secuencias de palabras (Miller *et al.*, 2012). En trabajos recientes se ha optado por crear medidas híbridas a partir de alguna combinación de las opciones antes mencionadas. Estas medidas han dado mejores resultados al ser combinadas que al ser utilizadas individualmente por lo que hay evidencia de que las medidas híbridas mejoran el funcionamiento de los sistemas (Agirre *et al.*, 2013).

Estas combinaciones se llevan a cabo mediante mezclas lineales o aditivas de las diferentes medidas existentes para comparar textos, lo cual no siempre resulta ser la opción idonea

Otro problema actual es que cada sistema de similitud textual está hecho en particular para una tarea. Ya sea detección de plagio, detección de paráfrasis, evaluación de resúmenes automáticos, etc. Por lo tanto, si se está trabajando en una tarea y se necesita cambiar de dominio, usualmente, se debe cambiar el método con el cual se

está detectando la similitud textual.

Por esta razón, en este trabajo se proponen dos puntos de vista diferentes para combinar medidas de similitud textual

1. Existen métodos que por sí mismos ya se encargan de comparar la similitud en secuencias de caracteres y al mismo tiempo verificar los caracteres compartidos por ambos textos. Estos algoritmos son usados principalmente para comparar secuencias biológicas como el ADN. Se propone adecuar estos algoritmos para que también, de forma interna, puedan hacer comparaciones semánticas entre los elementos de cada par de secuencias y así sean capaces de comparar textos; particularmente textos cortos.

2. Para tratar el problema de la particularidad de los métodos de similitud textual en dependencia de la tarea para la que están trabajando, se propone crear un generador automático de funciones. Para esto se planea utilizar un conjunto de medidas (algunas ya existentes y otras propuesta en este mismo trabajo) y construir mediante un programa genético un ensamblador de estas medidas para que, sin importar la tarea, encuentre una de las mejores relaciones para encontrar similitud textual.

1.2. Motivación

Como se mencionó al inicio de este capítulo, el procesamiento de lenguaje natural en general y la tarea de similitud textual de forma particular, tienen muchas tareas de gran impoortancia donde pueden ser aplicadas. Un avance significativo en esta área se traduciría en avances en tareas como: traducción automática o detección de plagio. Sin embargo, los modelos de similitud empleados hasta el día de hoy no han sido definitivos.

Por lo tanto, el desarrollo de uno o varios métodos de detección de similitud textual sigue siendo una línea de investigación abierta.

En esta tesis, se centrarán los esfuerzos en definir métricas de similitud textual especializadas en textos cortos. Esto con la intención de que sean aplicables en tareas como detección de plagio o detección de paráfrasis.

1.3. Objetivos

1.3.1. Objetivo general

Con base en lo antes mencionado, el objetivo general de esta tesis es:

> Diseñar e implementar nuevos métodos para calcular el grado de similitud entre dos textos cortos utilizando una mezcla entre traslape de conjuntos de palabras, similitud entre secuencias de información semántica extraída de ontologías o colecciones de documentos.

1.3.2. Objetivos específicos

Los objetivos específicos de esta tesis son los siguientes:

1. Extender algoritmos existentes para el cálculo de similitud o alineamiento de cadenas biólogicas para su aplicación en la tarea de detección de similitud textual enriqueciéndolos con información semántica proveniente de ontologías o de algún corpus.
2. Proponer un método supervisado, basado en programación genética, que genere, automáticamente funciones para detectar similtud textual

combinando medidas de tres diferentes categorías:

   a) Medidas que detectan el traslape de conjuntos de palabras

   b) Medidas que detectan la similitud de las secuencias de palabras

   c) Medidas que detectan inserciones o transformaciones semánticas

3. Adecuar los métodos de los puntos 1 y 2 para que puedan ser utilizados en la tarea de detección de paráfrasis y detección de plagio.

4. Evaluar los resultados del punto 3 para comprobar su funcionamiento.

1.4. Organización de esta tesis

La manera en que está organizado el contenido de este documento es la siguiente:

En el capítulo 2 se describe la tarea de detección de similitud textual. También se describen los trabajos más relevantes relacionados con esta línea de investigación. Se hace énfasis en los trabajos que utilizan medidas de traslape de conjuntos de palabras, similitud de secuencias e información semántica ya que son el centro de atención de esta tesis.

En el capítulo 3 se presentan los conceptos básicos requeridos para definir el problema de detección de similitud textual y para entender el resto del documento. También se incluye la definición de algoritmos utilizados en la propuesta de este trabajo como algoritmos de alineación de secuencias biológicas y programación genética, así como ejemplos de aplicaciones de los sistemas de detección de similitud textual.

En el capítulo 4 se presentan tres de los métodos propuestos en este trabajo. En este capítulo se describe cómo modificar dos algoritmos de alineación de secuencias biológicas y uno de distancia de edición enriqueciéndolos con información semántica para que sean capaces detectar similitud entre un par de textos.

En el capítulo 5 se presenta el otro método propuesto. En este capítulo se presenta la propuesta de crear automáticamente funciones para detectar similitud textual de forma supervisada a través de un algoritmo de programación genética combinando medidas que consideran traslape, orden y relación semántica de las palabras.

En el capítulo 6 se describen las colecciones utilzadas y se muestran los resultados experimentales obtenidos al evaluar el desempeño de los métodos propuestos y una comparación experimental contra los resultados reportados en el estado del arte sobre las mismas colecciones. También se presenta una pequeña discusión de los resultados.

Finalmente, en el capítulo 7 se exponen las conclusiones y algunas direcciones a seguir como trabajo futuro.

Capítulo 2

Tarea de detección de similitud textual

En este capítulo, primero se mencionará en qué consiste la tarea de detección de similitud textual. Posteriormente se hace un recuento de los principales modelos para atacar este problema. Se presentarán los trabajos más reelevantes en esta tarea; divididos en cuatro tipos que, como ya se ha mencionado antes, son de interés para este trabajo de tesis. Las categorías en que se clasifican y exponen estos métodos son:

1. Medidas basadas en secuencias de caracteres
2. Medidas basadas en traslape de conjuntos de palabras
3. Medidas basadas en información semántica
4. Medidas híbridas.

Finalmente, se expondrán dos tareas que utilizan la detección de sismilitud como principal motor. Estas tareas son la detección de paráfrasis y de plagio. Se describirá de

manera breve en qué consiste cada tarea y se exponen los principales enfoques que se han propuesto hasta el día de hoy.

2.1. Similitud textual

La tarea de similitud textual se encarga de comparar textos para conocer el parecido entre ellos. Los métodos de similitud textual han tenido aplicaciones muy importantes durante muchos años en los campos de lenguaje natural (Feng *et al.*, 2014) y recuperación de la información (Veselkov *et al.*, 2011).

Una variante de estos métodos son los que calculan el grado de similitud en textos cortos; una tarea un poco más complicada que la original por la poca información que se maneja para poder obtener el parecido de los textos (Sriram *et al.*, 2010).

Algunos ejemplos de aplicaciones donde se calcula la similitud en textos cortos son: Evaluación de traducción automática (Castillo & Estrella, 2012), detección de paráfrasis (Bhagat & Hovy, 2013), detección de plagio (Barrón-Cedeno *et al.*, 2010), evaluación de resúmenes automáticos (Oliva *et al.*, 2011), respuestas automáticas (Mohler *et al.*, 2011), etc.

A continuación se mencionarán algunos de los modelos formales que se han propuesto para detectar similitud textual.

2.2. Modelos de similitud

Hedrick (1975) fue de los primeros investigadores en abordar el problema de detectar el grado de similitud entre textos. Desde ese entonces hasta el día de hoy, se han desarrollado diversos modelos formales que hacen que la similitud sea un concepto bien

definido capaz de ser capturado en estos modelos formales.

En la siguiente sección se mencionarán dos modelos formales: Modelo geométrico (Shepard, 1962) y el modelo de teoría de conjuntos (Tversky, 1977), que posiblemente sean los modelos más populares utilizados hasta ahora para la detección de similitud textual.

2.2.1. Modelos formales de similitud

Como ya se mencionó, para esta discusión se tomaron en cuenta dos modelos formales: el geométrico y el de teoría de conjuntos. Para una discusión más exhaustiva se recomienda leer a Decock & Douven (2011).

Modelo geométrico

El modelo geométrico (Widdows, 2004) representa objetos en un espacio métrico $< X, \delta >$ logrando manipular el espacio geométrico de manera algebraica, donde cada objeto es representado por un punto $x \in X$ y $\delta(a, b)$ es la distancia entre un objeto a y un objeto b. Esta distancia es una función sobre X que retorna un número no negativo, donde, entre menos distancia se puede interpretar como mayor similitud. En un espacio geométrico, para todos los puntos $a,\ b,\ c \in X$ se cumplen los axiomas de una métrica: minimalidad, simetría e inequidad triangular (Tversky & Gati, 1982):

$$\textit{Minimalidad:} \qquad \delta(a, b) \geq \delta(a, a) = 0 \tag{2.1}$$

$$\textit{Simetría:} \qquad \delta(a, b) = \delta(b, a) \tag{2.2}$$

$$\textit{Inequidad triangular:} \qquad \delta(a, b) + \delta(b, c) \geq \delta(a, c) \tag{2.3}$$

Modelos de teoría de conjuntos

El modelo geométrico ha sido criticado por considerar a objetos como similares cuando no lo son. Por ejemplo, Cuba, es muy similar a Haití por su cercanía geográfica, mientras que Cuba es similar a Rusia por su afinidad política. Haití y Rusia no son para nada similares, ni en su ubicación geográfica ni en su afinidad política. El problema es que el axioma de inecuación triangular dice que $\delta(Haití, Cuba) + \delta(Cuba, Rusia) \geq \delta(Haití, Rusia)$, esto es que, la distancia entre Haití y Rusia no puede ser más grande que la suma de las distancias entre Haití y Cuba y Cuba y Rusia cuando sabemos que no es así (Tversky, 1977).

Por esta razón, Tversky (1977) propone utilizar teoría de conjuntos para calcular la similitud de dos objetos. El modelo se basa en las características de los objetos a comparar. La similitud $sim(a, b)$ se calcula a partir de una función que coteja los elementos que comparten tanto a, como b y tomando en cuenta, también, los elementos que solo están en a y en b, es decir:

$$sim(a, b) = f(a \cap b,\ a - b,\ b - a) \tag{2.4}$$

De esta forma, se toman en cuenta las características que comparten dos objetos pero, también se toman en cuenta, directamente, los elementos que hacen que se distinga a a de b.

2.3. Midiendo el grado de similitud textual

En la literatura se han propuesto diversas métricas y medidas para obtener el grado de similitud entre dos textos. Estas medidas pueden dividirse, generalmente, en dos grandes grupos: *medidas compuestas* y *medidas no compuestas*. Las medias compuestas

dividen los textos regularmente en palabras, y calculan la similitud uno a uno de cada parte de ambos textos para después llegar a un solo resultado de similitud. Por otro lado, las medidas no compuestas, primero representan los textos mediante algún modelo abstracto, por ejemplo, en vectores de alta dimensionalidad, para posteriormente, hacer la comparación bajo dicha representación.

También, típicamente, en la literatura se pueden encontrar clasificaciones por el tipo de recursos en los que se basan para obtener los resultados de similitud entre palabras, para luego obtener la similitud general de los textos. Encontrar similitud entre palabras es fundamental para la detección de similitud textual ya sea para comparar oraciones, párrafos o documentos completos.

Las palabras pueden ser similares de dos formas: **léxicamente y semánticamente**. Las palabras son léxicamente similares si las cadenas contienen secuencias de caracteres semejantes. Por otro lado, las palabras son semánticamente similares si se están refiriendo a lo mismo dentro del contexto, sin importar que no compartan secuencias de caracteres.

Los métodos que utilizan similitud léxica son los métodos basados en secuencia de caracteres. Los métodos de similitud basados en estas secuencias son de los más usados en la literatura por la simplicidad para implementarlos (Gomaa & Fahmy, 2013).

Otra forma de encontrar el parecido entre dos textos es simplemente tomando en cuenta el traslape de conjuntos palabras que comparten ambos textos. Estos métodos representan una gran ventaja por la efectividad que pueden llegar a tener en comparación con la rapidez del cálculo (Chaudhuri *et al.*, 2006). El problema de estas medidas es que no toman en cuenta el orden de las palabras a diferencia de las medidas anteriores.

Por otro lado, los métodos que utilizan similitud semántica se pueden dividir en dos grupos: métodos basados en corpus (Velasco, 2013) y métodos basados en conocimiento (Banea *et al.*, 2013). Las medidas basadas en corpus, como su nombre lo indica, utilizan

grandes colecciones de datos para obtener estadísticas de los textos y a partir de ellas se calcula el grado de similitud de los textos de entrada. Un ejemplo de las medidas basadas en corpus son las que utilizan la medida *tf-idf* (Zhang *et al.*, 2011). Las medidas basadas en conocimiento utilizan recursos léxicos y semánticos que contienen conocimiento humano de las palabras como diccionarios o tesaurus (Gabrilovich & Markovitch, 2007). Uno de los proyectos más populares disponibles es el de WordNet (Fellbaum, 1998).

Recientemente, Ho *et al.* (2010) y Tsatsaronis *et al.* (2010) proponen un nuevo grupo en esta clasificación. Este grupo es el de las medidas híbridas, es decir, las medidas que utilizan tanto secuencia de caracteres como corpus como conocimiento.

A continuación se hablarán de los métodos utilizados para onbtener la similitud entre un par de palabras. En esta sección se abordaran medidas basadas en caracteres y medidas basadas en información semántica, primero abordando las medidas basadas en corpus y posteriormente, las medidas basadas en conocimiento. Después se mencionarán los métodos para obtener similitud entre dos textos basandose en terminos y traslape de conjuntos de palabras y finalmente, se abordarán algunas de las medidas híbridas.

2.3.1. Medidas de similitud entre palabras

Como ya se mencionó, estas medidas calculan la similitud a partir de la proximidad de los caracteres de las palabras. Las medidas basadas en secuencias pueden dividirse en dos: las medidas de similitud **léxicas** y las medidas de similitud **semánticas** (Gomaa & Fahmy, 2013).

A continuación se describirán algunas medidas léxicas, posteriormente se abordarán las medidas semánticas.

Medidas léxicas

Subsecuencia común más larga. LCS por sus siglas en inglés, es un algoritmo que calcula la similitud de dos cadenas basado en la longitud de la cadena continua de caracteres más larga que exista entre ambas cadenas (Zesch *et al.*, 2013).

Algoritmo de Levenshtein. También conocido como distancia de edición. El resultado de este algoritmo dinámico es el número mínimo de operaciones requeridas para transformar una palabra en otra. Se entiende por operaciones de edición, una inserción, eliminación o la sustitución de un carácter (Chowdhury *et al.*, 2013). Existen otras generalizaciones de la distancia de Levenshtein, como la distancia de *Damerau-Levenshtein*, que consideran el intercambio de dos caracteres como una operación. En esta medida mientras el valor sea más alto, el valor de similitud es muy bajo.

Jaro-Winkler. Esta medida utiliza el número de caracteres que comparten ambas palabras, tomando en cuenta los caracteres que están en la misma posición y los que están transpuestos (Balsmeier *et al.*, 2014), con estos datos la medida está definida como:

$$sim_{jaro}(t_1, t_2) = \begin{cases} 0 & si \quad m = 0 \\ \frac{1}{3}\left(\frac{m}{|t_1|} + \frac{m}{|t_1|} + \frac{m-t}{m}\right) & si\ no \end{cases} \tag{2.5}$$

Donde:

- m es el número de caracteres que coinciden en su posición en ambas palabras, y t es el número de caracteres que coinciden y están en posiciones diferentes.

Similitud entre palabras considerando información semántica

Como ya se mencionó al principio de este capítulo, estas métricas, calculcan la similitud semántica entre dos palabras con la información obtenida a través de grandes volumenes de información. A continuación se abordarán algunos trabajos que utilizan esta técnica para obtener el grado de parecido de dos palabras.

Similitud basada en corpus

Hiperespacio análogo al lenguaje. HAL por sus siglas en inglés crea un espacio semántico a través de las co-ocurrencias de las palabras en un corpus. Palabra por palabra se contruye una matriz M donde M_{ij} representa qué tanto co-ocurrió la palabra i con la palabra j en la colección de datos. El valor en cada posición de la matriz M se calcula de forma acumulativa. Así, mientras un par de palabras obtenga un valor alto en la matriz, es evidencia de que se refieren a lo mismo dentro del contexto (Lund *et al.*, 1995; Kaufman *et al.*, 2013).

Análisis semántico latente. LSA, es posible que sea la técnica basada en corpus más popular. Esta técnica asume que las palabras que semánticamente son similares co-ocurrirán en pequeños pedazos del texto. Para capturar eso, se contruye una matriz donde los renglones representan los párrafos del textos y las columnas representan solo palabras. Posteriormente, como esta matriz regularmente es muy grande, se utiliza una técnica matemática llamada *descomposición en valores singulares* y es usada para reducir las dimensiones de la matriz tratando de mantener la similitud original. Al final las palabras se comparan de forma vectorial utilizando el coseno del ángulo que forman los vectores a comparar (Landauer *et al.*, 2013; Babcock *et al.*, 2014).

Información mutua - Recuperación de la información. Es un método para calcular similitud entre palabras. Esta técnica utiliza métodos avanzados de búsqueda de *queries* para calcular la probabilidad de que una palabra sea similar a otra. Esta probabilidad se calcula midiendo qué tan a menudo co-curren palabras cerca de otras en una colección de páginas web. El valor más alto es el resultado de este método (Chew, 2014).

Similitud de palabras basada en conocimiento

Los métodos de similitud basada en conocimiento son métricas que se basan en encontrar la similitud entre palabras utilizando información derivada de redes semánticas (Zhu *et al.*, 2014). WordNet es la red semántica más popular hasta el día de hoy (Rada *et al.*, 1989; Pal & Saha, 2014).

WordNet es una gran base de datos léxica en inglés. Sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos cognitivos (synsets), cada uno expresando un concepto distinto. Los synsets están vinculados entre sí por medio de las relaciones conceptuales semánticas y léxicas (Miller, 1995).

Básicamente existen seis funciones de similitud semántica basadas en conocimiento (en WordNet):

- Resnik (**res**) (Resnik, 1995; Ştefănescu *et al.*, 2014)
- Lin (**lin**) (Lin, 1998; Bordes *et al.*, 2014)
- Jiang y Conrath (**jcn**) (Jiang & Conrath, 1997; Zheng & Cai, 2014)
- Leacock y Chodorow (**lch**) (Leacock & Chodorow, 1998; Abdalgader, 2014)
- Wu y Palmer (**wup**) (Wu & Palmer, 1994; Surianarayanan *et al.*, 2014)
- *Path length* (**path**) (Budanitsky & Hirst, 2006; Song *et al.*, 2014)

Resnik Devuelve un valor continuo que denota qué tan similares son dos palabras, indicando que entre mayor sea el valor devuelto mayor es la similitud entre las palabras. Este cálculo esta basado en el contenido de información en la red, es decir, a partir del nodo en común más cercano a las palabras que se están comparando. Esta función es en su totalidad dependiente de cómo se construyó la red de conocimiento por lo que cualquier cambio en la colección hace que el resultado de similitud entre dos palabras pueda cambiar de forma abrupta. Resnik se define como:

$$sim = IC(lcs) \tag{2.6}$$

Donde $IC(lcs)$ es la profundidad del nodo donde se encuentra el ancestro más cercano en común.

Lin Retorna un valor continuo denotando la similitud entre un par de palabras y el ancestro más cercano y que sea común entre ellas. La relación de este método está definida como:

$$sim = \frac{2 * IC(lcs)}{IC(s1) + IC(s2)} \tag{2.7}$$

Donde, $IC(s1)$ es la profundidad del nodo en la red donde se encuentra la palabra uno. $IC(s2)$ es la profundidad del nodo en la red donde se encuentra la palabra dos. Finalmente $IC(lcs)$ es la profundidad del nodo donde se encuentra el ancestro más cercano en común.

Jiang y Corath Retorna un valor continuo denotando la similitud entre un par de palabras . Este método está basado en las palabras que se van a comparar y el ancestro más cercano a las dos palabras. El cálculo de esta función está definido como:

$$sim = \frac{1}{IC(s1) + IC(s2) - 2 * IC(lcs)} \tag{2.8}$$

Donde, $IC(s1)$ es la profundidad del nodo en la red donde se encuentra la palabra uno. $IC(s2)$ es la profundidad del nodo en la red donde se encuentra la palabra dos. Finalmente $IC(lcs)$ es la profundidad del nodo donde se encuentra el ancestro más cercano en común.

Leacock y Chodorow De este método se obtiene un valor que denota el grado de similitud entre dos palabras que se están comparando. Esta medida está basada en el cálculo de la ruta más corta entre las palabras en la red, pero, tomando en cuenta la profundidad máxima de las palabras. Para esta medida se toma en cuenta qué tan lejos se encuentra el ancestro común más cercano de las palabras comparadas. La función está definida como:

$$sim = -\log\left(\frac{p}{2d}\right) \tag{2.9}$$

Donde p es la longitud del camino más corto entre las palabras y d es la profundidad máxima de las palabras en la red.

Wu y Palmer Muy similar a la medida anterior, también se calcula la ruta más corta entre las palabras comparadas. La diferencia recae en el uso de uno de los ancestros de las palabra en la red. En la medida anterior se utiliza el ancestro en común más cercano; en esta medida se utilizará el ancestro más cercano a alguna de las dos palabras. Esta medida se define como:

$$sim = \frac{2 * Profundidad(LCS)}{Profundidad(IC(s1)) + Profundidad(IC(s2))} \tag{2.10}$$

Path length Devuelve un valor continuo que denota qué tan similares son dos palabras, con base en la trayectoria más corta que conecta las palabras en el árbol obtenido únicamente a través de relaciones del tipo *es-un*. El resultado está acotado en el rango de cero a uno, donde cero significa nula similitud y uno significa máxima similitud. Esta medida se define como:

$$sim = SP(IC(s1), IC(s2)) \tag{2.11}$$

Donde, $SP()$ es la función que devuelve la longitud del camino más corto de dos palabras en el árbol de *WordNet*.

2.3.2. Medidas de similitud entre textos

Basadas en el modelo geométrico

Distancia euclidiana Se define como la distancia más corta entre dos puntos. Para utilizar esta distancia, los textos se deben representar en un espacio euclidiano mediante vectores, donde cada dimensión de los vectores representa una palabra de los textos (Huang, 2008). La distancia euclidiana está definida como:

$$dis_e(t_1, t_2) = \sqrt{\sum_{i=1}^{n}(t_{1i} - t_{2i})^2} \tag{2.12}$$

Donde:

- n es la cardinalidad del vocabulario existente.
- t_1 y t_2 son los vectores.
- t_{1i} y t_{2i} son los i-ésimos elementos en los vectores que representan t_1 y t_2 respectivamente.

Distancia manhattan La métrica usual de la geometría euclideana es reemplazada por una nueva métrica en la cual la distancia entre dos puntos es la suma de las diferencias (absolutas) de sus coordenadas utilizando los mismos vectores que en la medida anterior (Wajeed & Adilakshmi, 2011). Esta distancia está definida como:

$$dis_m(t_1, t_2) = \sum_{i=1}^{n} |t_{1i} - t_{2i}| \tag{2.13}$$

Basadas en teoría de conjuntos

Coeficiente de Dice Este coeficiente se basa en la teoría de conjuntos. Toma el número de las palabras que comparten ambas cadenas y los divide entre el número total de la suma de las palabras del texto uno y dos (Al-Shamri, 2014). Es decir:

$$sim_D(t_1, t_2) = 2\frac{|t_1 \cap t_2|}{|t_1| + |t_2|} \tag{2.14}$$

Esta métrica está normalizada entre cero y uno donde cero es nula similitud mientras que uno se refiere a la máxima similitud.

Coeficiente de Jaccard Parecido al coeficiente anterior, este coeficiente se obtiene al dividir la intersección de términos entre la unión de los mismos (Cheng-Hui *et al.*, 2011), Es decir:

$$sim_J(t_1, t_2) = \frac{|t_1 \cap t_2|}{|t_1 \cup t_2|} \tag{2.15}$$

N-gramas Los n-gramas son subsecuencias de n palabras tomados de las cadenas de textos a comparar. El cálculo de similitud mediante n-gramas consiste en dividir el número de n-gramas que comparten ambas cadenas entre el número total de n-gramas

tal y como lo hace el coeficiente de Jaccard pero esta vez sobre n-gramas (Barrón-Cedeno *et al.*, 2010). Es decir:

$$sim_{ngramas}(t_1, t_2) = \frac{|ngramas(t_1) \cap ngramas(t_2)|}{|ngramas(t_1) \cup ngramas(t_2)|} \tag{2.16}$$

Coeficiente de traslape Similar al coeficiente de Jaccard pero considera solo la cardinalidad de caracteres del texto más pequeño en lugar de la únion de los caracteres (Gomaa & Fahmy, 2013). El coeficiente se define de la siguiente manera:

$$sim_T(t_1, t_2) = \frac{|t_1 \cap t_2|}{min(|t_1|, |t_2|)} \tag{2.17}$$

Coeficiente de coseno Este coeficiente se obtiene diviendo la cardinalidad de la unión de los dos conjuntos entre la raíz cuadrada del producto de las cardinalidades de los conjuntos considerados. Este coeficiente se define como:

$$sim_C(t_1, t_2) = \frac{|t_1 \cap t_2|}{\sqrt{|t_1||t_2|}} \tag{2.18}$$

2.3.3. Métodos híbridos

Existen diversas formas para obtener el grado de similitud de cada par de palabras entre dos textos. Zesch & Gurevych (2010) hacen un resumen de algunos de los métodos existentes para obtener el grado de similitud semántica a nivel de palabras. Este resumen incluye las medidas de Wu & Palmer (1994), Lesk (1986), Leacock & Chodorow (1998), Jiang & Conrath (1997) y Hirst & St-Onge (1998). A partir de estas medidas, muchos trabajos se han basado para enriquecerlas y obtener mejores resultados.

Ahora se explicarán algunos de los trabajos que utilizan la similitud de palabras como motor para su propuesta.

Mihalcea *et al.* (2006)

En este trabajo se propone una comparación semántica a la vez de cada par de palabras en ambos textos para después utilizar un amplio conjunto de medidas en combinación con una estrategia de agregación bidireccional. Las medidas propuestas son *PMI-IR* (Kaur & Hornof, 2005), análisis semántico latente (Landauer *et al.*, 1998), Leacock & Chodorow (1998), Lesk (1986), Wu & Palmer (1994), Resnik (1995), Lin (1998) y Jiang & Conrath (1997).

En un siguiente trabajo, en (Mohler & Mihalcea, 2009), hacen una extensión al trabajo original agregando las medidas del camino más corto en la taxonomia de *WordNet* (Rada *et al.*, 1989) y la medida de Hirst & St-Onge (1998).

La propuesta de agregación calcula la similitud del texto uno hacia el texto dos y viceversa maximizando el grado de similitud de cada par de palabras. La similitud de palabras se pondera por el *idf* de cada palabra (Chowdhury, 2010). Finalmente, el resultado es calculado por el promedio de ambas direcciones de similitud de la siguiente manera:

$$sim(t_1, t_2) = \frac{1}{2}\left(\frac{\sum_{w \in t_1} maxSim(w, t_2) idf(w)}{\sum_{w \in t_1} idf(w)} + \frac{\sum_{w \in t_2} maxSim(w, t_1) idf(w)}{\sum_{w \in t_2} idf(w)}\right) \quad (2.19)$$

Donde, t_1 y t_2 son los dos textos a comparar.

Zini *et al.* (2006)

En este trabajo se propone un método con el fin de detectar plagio. Para esto el método propuesto obtiene el grado de similitud entre el texto fuente y el texto sospechoso.

Primero, se obtiene una estructura de árbol de cada documento. Se definen tres

niveles donde el nivel cero o las hojas son las palabras del documento. El nivel uno son las frases, el nivel dos los párrafos y por último, el documento completo es la raíz. El autor en este trabajo se refiere tanto a palabras, frases y párrafos como *chunks*. De esta forma c_i^L denota al *chunk* del nivel L del documento i. Así

- $\lambda(c)$ el número de sub-*chunks* que contiene c
- $|c|$ es el número de caracteres en c
- $\sigma(c^L)$ el número de sub-*chunks* del nivel $L-1$ en c^L

Con estas funciones dadas se define la similitud estructural como:

$$\zeta(c_i^L, c_j^L) = 1 - \frac{ed(c_i^L, c_j^L)}{max(\lambda(c_i^L), \lambda(c_j^L))} \tag{2.20}$$

Donde $ed()$ es la distancia de edición entre dos textos.

Finalmente, la función de similitud queda definida como:

$$\xi(c_i^L, c_j^L) = \frac{\sum w(c_i^{L-1}, c_j^{L-1}) P(c_i^{L-1}, c_j^{L-1})}{\sum w(c_i^{L-1}, c_j^{L-1})} \tag{2.21}$$

Donde:

$$w(c_i^{L-1}, c_j^{L-1}) = max\left(\frac{|c_i^{L-1}|}{|c_i^L|}, \frac{|c_j^{L-1}|}{|c_j^L|}\right) \tag{2.22}$$

y $P()$ está definida como:

$$P(c_i^{L-1}, c_j^{L-1}) = \begin{array}{ll} \frac{\alpha_L \xi(c_i^L, c_j^L) + \beta_L \zeta(c_i^L, c_j^L)}{\alpha_L + \beta_L} & si \quad L > 0 \\ \zeta(c_i^L, c_j^L) & si \quad L = 0 \end{array} \tag{2.23}$$

α_L y β_L son parámetros de peso ajustables.

Liu *et al.* (2007)

En este trabajo se reporta un método que utiliza información semántica y partes del discurso para obtener el grado de similitud entre dos frases. Para esto se define una extensión al algoritmo original de Dynamic Time Warping (Islam *et al.*, 2014) para que acepte información semántica proveniente de WordNet. Después divide la frase en dos frases más pequeñas, una donde contiene solo verbos y predicados y otra donde contiene el resto de la frase original. Se aplica por separado el método propuesto a cada clase y finalmente se hace un promedio ponderado haciendo preferencia a la frase de verbos y predicados.

Su *et al.* (2008)

Este método utiliza una fusión entre los algoritmos de distancia de edición y una simplificación del algoritmo de *Smith-Waterman*.

Primero aplica el algoritmo tradicional de Levenshtein. El resultado de este algoritmo es una matriz donde se muestra el camino que siguió el algoritmo para obtener la distancia entre los textos. De acuerdo con los autores de este trabajo, la máxima similitud se concentrará alrededor de la diagonal principal del resultado del algoritmo de distancia de edición.

El algoritmo de distancia de edición generalmente parte la matriz en lo que se podría considerar la parte similar entre los textos y la parte que no lo es. Con esta información el autor aplica el algoritmo de Smith-Waterman simplificado sobre la misma porción que es similar en la matriz resultante de la distancia de edición.

Jimenez *et al.* (2010)

Este es un trabajo que también utiliza WordNet para obtener similitud entre pares de palabras. Utiliza como base la medida Jaccard para medir el traslape de conjuntos palabras. La diferencia es que implementa una función para re-calcular la cardinalidad de los conjuntos de palabras.

Generalmente, si el conjunto $s = \{gato, leon, puerta\}$, la cardinalidad de s se establece en 3. Pero en este caso, el autor asume que *leon* y *gato* no deberían contar como dos objetos totalmente distintos puesto que las funciones de similitud en WordNet indican que estas palabras son semánticamente cercanas, por lo que la cardinalidad se convierte en un valor real, en este caso sería 2.3.

Wang & Cer (2012)

En este trabajo se define una extensión al algoritmo de distancia de edición. Esta extensión consiste en constriur una máquina de estados finitos probabilística donde incluye tres estados, que representan las posibles operaciones que se pueden hacer en una distancia de edición (insertar, eliminar o sustituir).

De un corpus este algoritmo aprende la probabilidad de pasar de un estado a otro y posteriormente, con cadenas ocultas de Marcov obtiene el grado de probabilidad de convertir una cadena a otra, que finalmente, se convertirá en el grado de similitud de ambos textos.

Castillo & Estrella (2012)

Este trabajo se realizó con la finalidad de evaluar sistemas de tradución automática. Este sistema básicamente extrae los valores de similitud de cada par de palabras utilizando las funciones definidas en WordNet (como **res**, **lin**, **jch**, etc.) para construir

vectores característicos. Así, este método entrena y prueba con el algoritmo de máquinas de vectores de soporte y se definen umbrales para determinar la evaluación de cada sistema.

Bär *et al.* (2012)

En este trabajo se define un algoritmo de aprendizaje. Utiliza diversas medidas que calculan similitud para mezclarlas mediante un algoritmo de regresión lineal y así obtener el grado de similitud final.

Algunas de las medidas que utiliza este método son: subsecuencia común más larga, n-gramas de caracteres y palabras, Jiang y Corath, Lin, Resnik, análisis semántico explícito (Egozi *et al.*, 2011), vinculación textual (Bos, 2014), entre otras. En total utiliza 27 medidas para obtener el resultado final.

2.4. Algunas aplicaciones de detección de similitud textual

La tarea de detección de similitud textual, en la mayoría de las ocasiones no es una tarea final, más bien, es un proceso intermedio para resolver una tarea más específica. Obtener el grado de similitud entre un par de textos nos da noción del parecido que existe entre ellos pero también aporta información para poder tomar decisiones de acuerdo al resultado obtenido.

Existen diversas tareas que calculan el grado de similitud entre dos textos para obtener un resutado final. Ejemplos de estas tareas son la detección de paráfrasis y la detección de plagio. Estas tareas son las que aborda esta tesis para evaluar los métodos propuestos en los capítulos 4 y 5. En el capítulo 6 se mostrarán los resultados obtenidos

de este trabajo en estas tareas.

A continuación se describirá un poco en qué consiste cada una de estas tareas y se describirán los trabajos más sobresalientes en el estado del arte, que finalmente serán los métodos con los que se compararán los resultados obtenidos.

2.4.1. Detección de paráfrasis

La paráfrasis es una operación intelectual que consiste en trasladar con nuestras propias palabras las ideas que ha expresado de manera oral o escrita otra persona, con el fin de sustituir la información a un lenguaje más personalizado y lograr una mejor comprensión (Fader *et al.*, 2013).

Lo más común al momento de parafrasear es sustituir sólo algunas palabras por sinónimos y cambiar el orden de algunas frases sin alterar su significado. La otra manera de parafrasear es reconstruyendo la redacción de una manera más resumida, pero sin perder la esencia de la información o del mensaje original (Cajías, 2014).

Existen dos técnicas principales para generar una paráfrasis:

- Paráfrasis mecánica: Consiste en sustituir por sinónimos o frases alternas las expresiones que aparezcan en un texto, con cambios sintácticos mínimos.
- Paráfrasis constructiva: Consiste en la reelaboración del enunciado, dando origen a otro con características muy distintas, pero conservando el mismo significado.

La tarea de detección de paráfrasis consiste en, dado un par de textos, decidir si uno de los textos es o no paráfrasis del otro. En general la tarea de detección de paráfrasis es una decisión de tipo binaria, pero se puede extender para que en lugar de generar salidas de tipo sí o no, también genere salidas indicando el tipo de técnica de paráfrasis que se utilizó para llegar de un texto a otro.

2.4.2. Algunos enfoques para detectar paráfrasis

Existen diversos enfoques para resolver la tarea de detección de paráfrasis. En este trabajo se abordarán los dos trabajos con mejores resultados en el estado del arte que serán los trabajos con los que finalmente se comparará la propuesta de esta tesis. Si se desea profundizar sobre trabajos especializados en detección de paráfrasis se recomienda visitar el sitio del estado del arte sobre el corpus de detección de paráfrasis de *Microsoft*[1]

Fernando & Stevenson (2008)

En este trabajo se aborda el problema de detección de paráfrasis de manera no supervisada. Consiste en construir una matriz W a partir de la similitud semántica resultante de comparar palabra por palabra en el par de textos. Para obtener dicha información semántica los autores de este trabajo utilizaron *WordNet*. El grado de similitud entre ambos textos se obtiene de la siguiente manera:

$$sim(a, b) = \frac{\vec{a} W \vec{b}^T}{|\vec{a}||\vec{b}|} \tag{2.24}$$

Donde $\vec{a}$ y $\vec{b}$ son los vectores con los valores $Tf - Idf$ de cada palabra de los textos a y b respectivamente.

Finalmente, si el grado de similitud sobrepasa un umbral se dice que los textos son paráfrasis, de lo contrario, la respuesta es negativa.

Madnani *et al.* (2012)

En este trabajo, también se aborda el problema de detección de paráfrasis pero desde un punto de vista supervisado. Este método utiliza tres clasificadores y mediante un pro-

[1]Visto el 15 de octubre de 2014
http://aclweb.org/aclwiki/index.php?title=ParaphraseIdentification(Stateoftheart)

medio del resultado de los tres obtiene el veredicto final. Este sistema utiliza: Clasificador de regresión logística, máquinas de vectores de soporte y k vecinos más cercanos.

Para cada uno de estos clasificadores, los autores proponen utilizar como atributos diversas técnicas de evaluación de traducción automática como:

- BLEU (Papineni *et al.*, 2002). Esta es la medida más utilizada para evaluar traducciones automáticas. Esta medida esta basada en el traslape de n-gramas entre ambos textos a comparar con diferentes valores de n.

- NIST (Doddington, 2002). Es una variante de BLEU, también funciona con n-gramas, pero, ésta obtiene un promedio aritmético de los n-gramas compartidos entre el total de n-gramas y luego uno geométrico para finalmente combinar los resultados.

- TER (Snover *et al.*, 2006) Es una distancia de edición que retorna el mínimo número de operaciones necesarias para hacer idéntica la traducción a evaluar y la traducción ideal. Las operaciones permitidas por esta distancia de edición son: Insertar, eliminar y sustituir.

- TERp (Snover *et al.*, 2009) Es una extensión de la medida anterior. Esta medida agrega operaciones basadas en *stemming*.

- METEOR (Denkowski & Lavie, 2010) Esta medida está basada en n-gramas al igual que BLEU pero ésta toma en cuenta tanto precisión como recuerdo, a diferencia de BLEU que solo toma en cuenta precisión. También lleva a cabo un preprocesamiento donde utiliza *stemming* y sinonimia.

- SEPIA (Habash & Elkholy, 2008) En este trabajo, el autor propone utilzar n-gramas estructurales los cuales son capaces de capturar mayor información que

los n-gramas tradicionales. Con el conjunto de n-gramas estruturales este método funciona igual que BLEU.

- MAXSIM (Chan & Ng, 2008) Este método trata el problema como cotejo de grafos bipartitos emparejando cada palabra de un texto con la más similar en el otro texto.

2.4.3. Detección de plagio

En la RAE se define plagio como "la acción copiar en lo sustancial obras ajenas, dándolas como propias". Eso es, básicamente copiar ideas. Con el crecimiento de la información en la red, se ha vuelto cada vez más fácil plagiar obras de cualquier índole, sobre todo, escritas. Por esta razón, ha surgido la necesidad de contar con herramientas confiables para detectar plagio en textos.

Existen dos tipos de detección de plagio en texto. La detección de plagio intrínsico y extrínseco.

La detección de plagio intrínsico consiste en, dado un documento, encontrar los fragmentos de texto que posiblemente fueron plagiados de alguna fuente sin conocimiento de otra cosa más que del propio texto (Zu Eissen & Stein, 2006).

La detección de plagio extrínseco consiste en, dado un par de textos, donde uno es el texto fuente y otro es el texto sospechoso, determinar si el documento sospechoso es o no plagio del original (Alzahrani & Salim, 2010). De ahora en adelante, cuando se mencione la tarea de detección de plagio, se estará haciendo referencia únicamente a la tarea de detección de plagio extrínseco.

2.4.4. Algunos enfoques para detectar plagio

Existen diversos enfoques para resolver la tarea de detección de plagio. Al igual que en la tarea de detección de paráfrasis, en esta sección se abordarán los dos trabajos con

mejores resultados en el estado del arte que serán los trabajos con los que finalmente se comparará la propuesta de esta tesis. Si se desea profundizar sobre trabajos especializados en detección de plagio se recomienda visitar el sitio del foro internacional PAN-CLEF [2]

Chong *et al.* (2010)

En este trabajo se propone un método para detección de plagio de manera supervisada Los autores proponen una serie de combinaciones entre técnicas de procesamiento de lenguaje natural para preprocesar cada par de textos; y 4 diferentes formas para que. tanto el documento fuente como el documento sospechoso sean comparados.

Se combinan técnicas como: lematización, separar el texto en palabras, etc. Y comparando los textos resultantes mediante:

1. Teoría de conjuntos.

2. Modelo de lenguaje probabilístico

3. Subsecuencia común más larga

4. Coincidencia de relaciones de dependencia

Al combinar cada técnica de comparación con los conjuntos de preprocesamiento, se construye un vector característico por cada par de documentos.

Como clasificador se utiliza el algoritmo de Naïve Bayes, que es alimentado en la fase de entrenamiento por un conjunto de vectores característicos resultantes de la comparación de diversos documentos.

[2]Visto el 15 de octubre de 2014
http://www.uni-weimar.de/medien/webis/research/events/pan-13/pan13-web/about.html

Sánchez-Vega *et al.* (2013)

En este trabajo, los autores definen cinco maneras diferentes de llevar a cabo el proceso de plagio:

1. Copiar completamente un texto (o porción del texto) sin hacer cambios.
2. Hacer pocas inserciones o eliminaciones de palabras.
3. Hacer muchas insercoines o eliminaciones de palabras.
4. Hacer pocas permutaciones de palabras.
5. Hacer muchas permutaciones de palabras.

Para cada una de las diferentes formas que los autores definen, se diseñó una máquina de Turing (Rendell, 2014) capaz de detectar el plagio en un fragmento de texto definido por ventanas en el texto completo. En las transiciones de las máquinas de Turing se va acumulando el valor de probabilidad de que el fragmento de texto sea o no plagiado. Finalmente, si la máquina de Turing se detuvo en un estado no final, el método retorna un cero. Por otro lado, si la máquina de Turing termina en un estado de aceptación, retorna el valor acumulado de probabilidad. El valor final resultante es el valor mayor entre las cinco máquinas de Turing.

Capítulo 3

Fundamentos

En este capítulo se presentan algunas nociones recomendadas al lector para poder entender mejor las propuestas de este trabajo de investigación.

Primero, se abordará la descripción de algunos algoritmos que se encargan de alinear secuencias biológicas. Estos algoritmos son muy importantes puesto que son la base de los métodos propuestos en el capítulo 4.

Posteriormente, se describirá en qué consiste un proceso de clasificación supervisada. Esto se debe a que en el capítulo 5, el método que se describe utiliza un proceso de este tipo para su funcionamiento.

Finalmente, se hace una breve descripción de lo que son los algoritmos evolutivos y su clasificación hasta llegar a la sección de programación genética, que es la base de la propuesta del capítulo 5.

3.1. Alineación de secuencias biológicas

Un alineamiento de secuencias en bioinformática es una forma de representar y comparar dos o más secuencias o cadenas de ADN, ARN o estructuras primarias proteicas

para resaltar sus zonas de similitud, que podrían indicar relaciones funcionales o evolutivas entre los genes o proteínas consultados. Las secuencias alineadas se escriben con las letras (representando aminoácidos o nucleótidos) en filas de una matriz en las que, si es necesario, se insertan espacios para que las zonas con idéntica o similar estructura se alinien (Langmead *et al.*, 2009).

Existen dos formas de alinear dos secuencias biolólogicas: *globales* y *locales*. Los alineamientos globales, que intentan alinear cada residuo de cada secuencia, son más útiles cuando las secuencias del problema inicial son similares y aproximadamente del mismo tamaño (no quiere decir que los alineamientos globales no puedan terminar en huecos). Una estrategia general de alineamiento global es el algoritmo **Needleman-Wunsch** basado en programación dinámica.

Por otro lado, los alineamientos locales son más útiles para secuencias diferenciadas en las que se sospecha que existen regiones muy similares. El algoritmo **Smith-Waterman** es un método general de alineamiento local basado en programación dinámica. Con secuencias suficientemente similares, no existe diferencia entre alineamientos globales y locales.

```
Global FTFTALILLAVAV
       F--TAL-LLA-AV

Local  FTFTALILL-AVAV
       --FTAL-LLAAV--
```

Figura 3.1: Ejemplo de la diferencia entre alineación global y local para las secuencias FTFTALILLAVAV y FTALLLAAV

En la figura 3.1 se muestra el resultado de alinear de manera global y local los pares de secuencias FTFTALILLAVAV y FTALLLAAV. En el alineamiento global se puede observar la tendencia a tratar de "estirar" la secuencia dos a lo largo de toda la secuencia uno para tratar de cubrir la mayor parte de la secuencia. Por otro lado, en el alineamiento

local se nota que la secuencia dos es partida en dos secuencias más cortas para cubrir pequeas zonas de la secuencia uno.

A continuación se describirán los algoritmos de Needleman-Wunsch y de Smith-Waterman.

3.1.1. Algoritmo Needleman-Wunsch

El algoritmo de Needleman-Wunsch sirve para realizar alineamientos globales de dos secuencias biológicas. Para que este algoritmo funcione necesita cierta información de la similitud de los caracteres que está utilizando y se representa en forma de tabla o matriz. Los biólogos generan esta tabla según la conveniencia del problema (Lan *et al.*, 2014). Por ejemplo, en esta tarea se trabaja con secuencias muy grandes con los caracteres A, G, C, T, y algún biólogo decide crear su tabla informativa como se indica en la figura 3.2. En esta tabla se puede ver que alinear el caracter A con otro caracter A es preferible que alinear un par de G y que lo que menos se desea es alinear un caracter G con un caracter C. Hasta el día de hoy existen muchas formas de crear estas tablas para la alineación de ADN (Jayaprada *et al.*, 2014).

$-$	A	G	C	T
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

Figura 3.2: Ejemplo de la tabla de información para alineación de cadenas de ADN

Este algoritmo dispone de tres operaciones disponibles para llevar a cabo el alineamiento global. Este algoritmo tiene la posibilidad, por cada par de caracteres en las secuencias a alinear, dejar un hueco en la primer secuencia, dejar un hueco en la segunda secuencia o finalmente, alinear el par de caracteres. También se define el costo de no

alinear algún caracter como un parámetro del algoritmo.

Definiendo como parámetros las secuencias a y b, S la matriz informativa y el d costo de dejar un hueco en cualquiera de las dos secuencias, formalmente, este algoritmo está definido como:

Caso base:

$$NW_{a,b}(i,0) = d * i$$
$$NW_{a,b}(0,j) = d * j$$

Algoritmo de Needleman-Wunsch:

$$NW_{a,b}(i,j) = max \begin{cases} NW_{a,b}(i-1,j) + d & \text{Hueco en secuencia a} \\ NW_{a,b}(i,j-1) + d & \text{Hueco en secuencia b} \\ NW_{a,b}(i-1,j-1) + S(a(i),b(j)) & \text{Alinear } a_i \text{ con } b_j \end{cases} \quad (3.1)$$

En el algoritmo 1 se define este procedimiento completo.

3.1.2. Algoritmo de Smith-Waterman

Este algoritmo se utiliza para llevar a cabo alineamientos locales de dos secuencias biológicas. Para esto, al igual que el algoritmo anterior, Smith-Waterman utiliza una matriz informativa para indicar los carcateres que se prefiere alinear y los caracteres que definitivamente no se desea alinear.

Este algoritmo, a diferencia de Needleman-Wunsch, cuenta con cuatro posibles operaciones. Estas operaciones permiten dejar un hueco en la secuencia uno, dejar un hueco

Algoritmo 1 Algoritmo de Needleman-Wunsch

Entrada: SecuenciaA, SecuenciaB, d, S // d es el costo de dejar espacios entre las secuencias y S es la tabla informativa

Salida: El grado de la mejor alineación global entre los textos A y B según la tabla S.

```
1: entero D[Len(SecuenciaA) + 1 ][Len(SecuenciaB) + 1], i, j, Alinear, DejarHuecoEnA,
   DejarHuecoEnB
2: para i = 0 hasta Len(SecuenciaA) hacer
3:    D[i][0] := i*d
4: fin para
5: para i = 0 hasta Len(SecuenciaB) hacer
6:    D[0][i] := i*d
7: fin para
8: para i = 1 hasta Len(SecuenciaA) hacer
9:    para j = 1 hasta Len(SecuenciaB) hacer
10:      Alinear := D[i-1][j-1] + S[SecuenciaA[i]][SecuenciaB[j]]
11:      DejarHuecoEnA := D[i-1][j] + d
12:      DejarHuecoEnB := D[i][j-1] + d
13:      D[i][j] := max(Alinear, DejarHuecoEnA, DejarHuecoEnB)
14:   fin para
15: fin para
16: devolver D[len(SecuenciaA)][len(SecuenciaB)]
```

en la segunda secuencia, alinear el par de caracteres que se está analizando en ese momento o finalmente terminar con la alineación local que se está llevando a cabo. Al igual que el algoritmo anterior, dejar un hueco en alguna de las dos secuencias tiene un costo.

Formalmente, donde a y b son las secuencias a alinear, S la matriz informativa y d el costo por dejar huecos, este algoritmo se define como:

Caso base

$$SW_{a,b}(i,0) = 0$$
$$SW_{a,b}(0,j) = 0$$

Algoritmo de Smith-Waterman:

$$SW_{a,b}(i,j) = max \begin{cases} 0 & \text{Corte del alineamiento local} \\ SW_{a,b}(i-1,j) + d & \text{Hueco en secuencia a} \\ SW_{a,b}(i,j-1) + d & \text{Hueco en secuencia b} \\ SW_{a,b}(i-1,j-1) + S(a(i),b(j)) & \text{Alinear } a_i \text{ con } b_j \end{cases} \tag{3.2}$$

En el algoritmo 2 se muestra el proceso completo de Smith-Waterman para obtener la mejor alineación local. El resultado de este algoritmo es la suma de la trayectoria mayor en la matriz resultante.

Algoritmo 2 Algoritmo de Smith-Waterman

Entrada: TextoA, TextoB, d

Salida: El costo de la mejor alineación local entre los TextoB y TextoA.

```
1: entero D[Len(TextoA) + 1 ][Len(TextoB) + 1], i, j, Alinear, DejarHuecoEnA, DejarHuecoEnB
2: para i = 0 hasta Len(TextoA) hacer
3:    D[i][0] := 0
4: fin para
5: para i = 0 hasta Len(TextoB) hacer
6:    D[0][i] := 0
7: fin para
8: para i = 1 hasta Len(TextoA) hacer
9:    para j = 1 hasta Len(TextoB) hacer
10:      Alinear := D[i-1][j-1] + S[SecuenciaA[i]][SecuenciaB[j]]
11:      DejarHuecoEnA := D[i-1][j] + d
12:      DejarHuecoEnB := D[i][j-1] + d
13:      D[i][j] := max(Alinear, DejarHuecoEnA, DejarHuecoEnB)
14:   fin para
15: fin para
16: devolver D
```

3.2. Aprendizaje supervisado

El aprendizaje supervisado es una técnica para deducir una función a partir de datos de entrenamiento. Los datos de entrenamiento consisten de pares de objetos (normalmente representados con vectores); una componente de este par son los datos que describen cada objeto y el otro, la clase a la que pertenece dicho objeto. La salida de la función puede ser un valor numérico o una etiqueta.

El objetivo del aprendizaje supervisado es crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber visto una serie de ejemplos, es decir, los datos de entrenamiento. Para ello, tiene que generalizar a partir de los datos presentados a las situaciones no vistas previamente (Moreno, 1994).

Con el fin de resolver un determinado problema de aprendizaje supervisado se tienen que considerar varios pasos:

1. Determinar el tipo de ejemplos de entrenamiento. Antes de hacer cualquier otra cosa, hay que decidir qué tipo de datos se van a utilizar para entrenar a un buen modelo de clasificación.

2. Reunir un conjunto de entrenamiento. El conjunto debe apegarse lo mejor posible a las características propias del mundo real. Se debe hacer cuidadosamente una selección de un conjunto de objetos de entrenamiento que se recopila junto con sus clasificaciones correspondientes.

3. Determinar el modelo de clasificación para generar una función aprendida que determinará las clases de los elementos de prueba. La precisión de la función aprendida depende en gran medida de cómo el objeto de entrada está representado. Normalmente, el objeto de entrada se transforma en un vector que contiene una serie de características que son descriptivas del objeto. El número de características no debe ser demasiado grande, pero debe ser lo suficientemente grande como para predecir con precisión la salida.

En la figura 3.3 se muestra cómo funciona en general un proceso de clasificación supervisada. Se puede ver que consta de dos partes importantes, a) de una fase de entrenamiento y b) de una fase de prueba. En la fase de entrenamiento se ve como dada una entrada y unas etiquetas (que también se les conoce como clases) se extraen las características de cada objeto en la colección de datos; posteriormente estas características con sus respectivas etiquetas son los parámetros que utiliza algún algoritmo de aprendizaje automático donde la salida de este algoritmo es un modelo de clasificación. Cuando el modelo está creado, éste está listo para clasificar datos nuevos a los cuales se les extraen las mismas características que se les extrajeron a los datos de entrenamiento. Finalmente el modelo obtendrá el resultado de la clasificación.

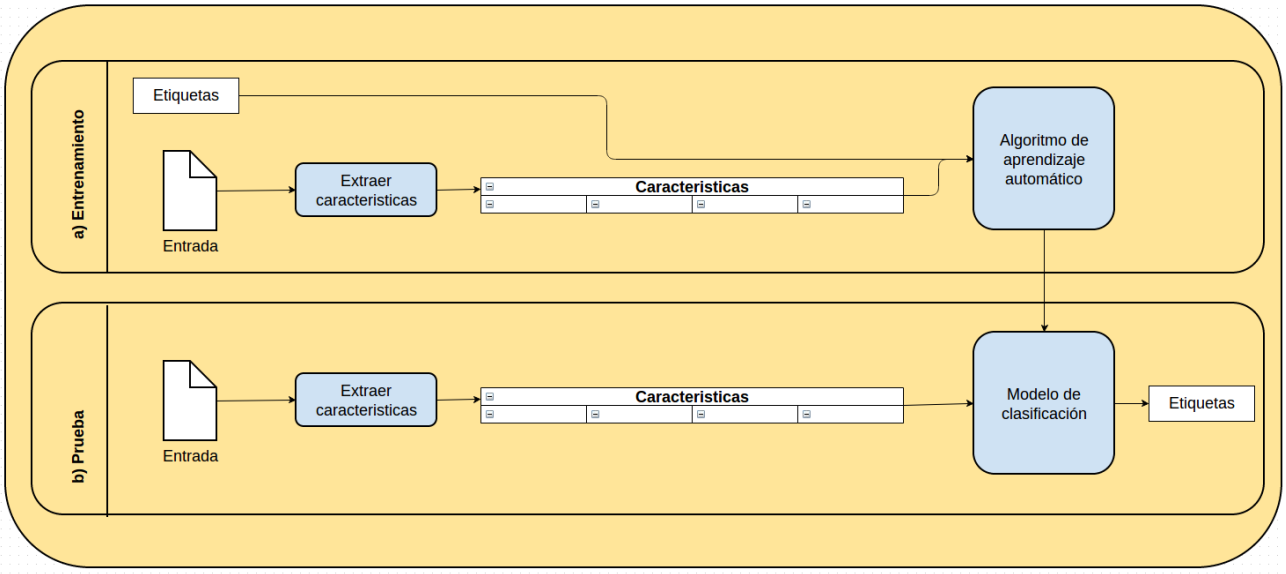

Figura 3.3: Representación gráfica del aprendizaje supervisado

Una amplia gama de algoritmos de aprendizaje automático están disponibles, cada uno con sus fortalezas y debilidades. Los clasificadores más utilizados son las redes neuronales (Liu *et al.*, 2014b), las máquinas de vectores de soporte (Araghinejad, 2014); el algoritmo de los k-vecinos más cercanos (Tomašev *et al.*, 2014), clasificadores bayesianos (Minnier *et al.*, 2014) y los árboles de decisión (Bui *et al.*, 2014). La relación clasificador-rendimiento depende en gran medida de las características de los datos que deben clasificarse. No hay una clasificación única que funciona mejor en todos los problemas dados.

Diversas pruebas empíricas se han realizado para comparar el rendimiento del clasificador y para encontrar las características de los datos que determinan el rendimiento del clasificador. La determinación de un clasificador adecuado para un problema dado, aún es más un arte que una ciencia (Soysal & Schmidt, 2010).

3.3. Algoritmos evolutivos

Los algoritmos evolutivos son métodos de optimización y búsqueda de soluciones basados en los postulados de la evolución biológica (Villamizar *et al.*, 2014). En ellos se mantiene un conjunto de entidades que representan posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando cada vez hacia mejores soluciones (Liao, 2010).

Los algoritmos evolutivos, y la computación evolutiva, son una rama de la inteligencia artificial. Son utilizados principalmente en problemas con espacios de búsqueda extensos y no lineales, en donde otros métodos no son capaces de encontrar soluciones en un tiempo razonable, evaluado mediante una función objetivo (Liau *et al.*, 2013). Una función objetivo es la función que será optimizada dadas las limitaciones o restricciones determinadas y con variables que necesitan ser minimizadas o maximizadas.

Siguiendo la terminología de la teoría de la evolución, las entidades que representan las soluciones al problema se denominan individuos o cromosomas, y el conjunto de éstos, población. Los individuos son modificados por operadores genéticos, principalmente el sobrecruzamiento, que consiste en la mezcla de la información de dos o más individuos; la mutación, que es un cambio aleatorio en los individuos; y la selección que consistente en la elección de los individuos que sobrevivirán y conformarán la siguiente generación. Dado que los individuos que representan las soluciones más adecuadas al problema tienen más posibilidades de sobrevivir, la población va mejorando gradualmente (Roy, 2013).

En la figura 3.4 se muestra el flujo que se lleva a cabo con un algoritmo evolutivo. Primero se genera una población inicial donde se evalua una función objetivo; apartir de este momento la población empieza a evolucionar con las operaciones disponibles. Esta operación se lleva a cabo hasta que los criterios de optimización son satisfactorios o hasta que se haya llegado a un límite de iteraciones.

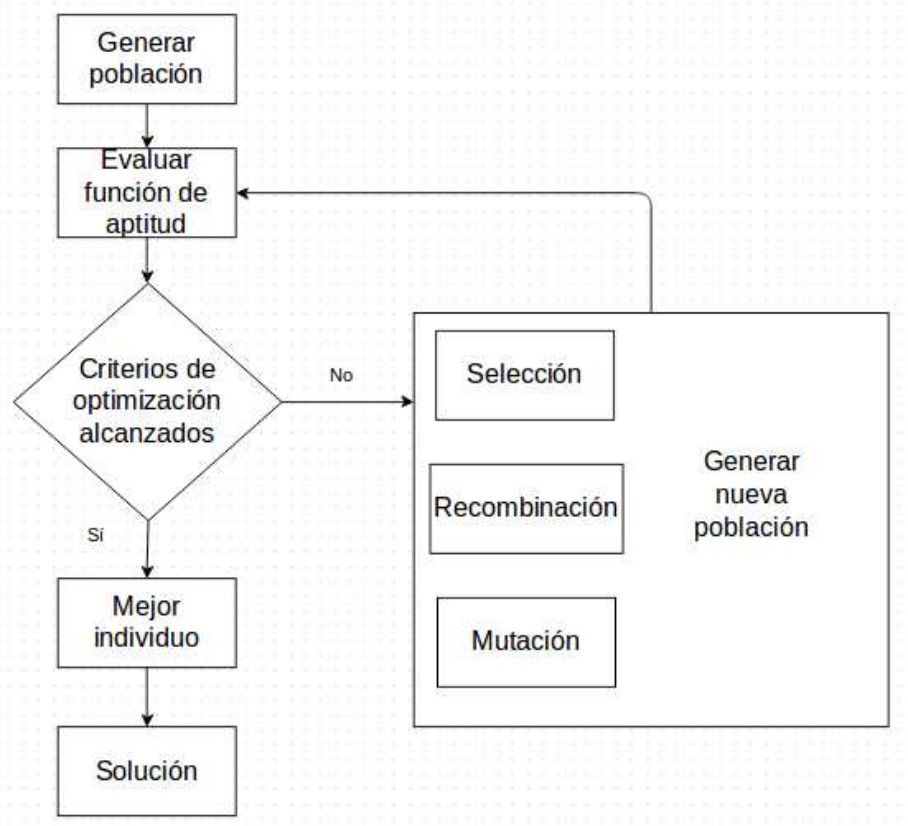

Figura 3.4: Representación gráfica de los algoritmos evolutivos

3.3.1. Paradigmas

Normalmente, se dice que existen tres paradigmas principales de algoritmos evolutivos: **Programación evolutiva**, **estrategias evolutivas** y **algoritmos genéticos**. Cada uno de estos paradigmas se originó independientemente y con distintas motivaciones. Actualmente, los algoritmos tienden a combinar características de estos tres y a incluir mecanismos de otros campos de estudio, tales como el aprendizaje automático, otros algoritmos de búsqueda, o diferentes estructuras de datos (Back *et al.*, 1997).

A continuación se habla brevemente de los algoritmos genéticos. A diferencia de los paradigmas de programación evolutiva y estrategias evolutivas, este paradigma se describe porque es el que se utilizará en el capítulo 5.

3.3.2. Algoritmos genéticos

Estos algoritmos hacen evolucionar a una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados (Goldberg, 1989).

Básicamente, el algoritmo genético funciona como sigue: en cada generación, se crea un conjunto nuevo de "criaturas artificiales" (cadenas) utilizando bits y partes más adecuadas del progenitor. Esto involucra un proceso aleatorio que no es, en absoluto, simple. La novedad que introducen los algoritmos genéticos es que explotan eficientemente la información histórica para especular sobre nuevos puntos de búsqueda, esperando un funcionamiento mejorado.

La forma más simple de algoritmo genético utiliza tres tipos de operadores: selección, cruce y mutación.

- Selección o reproducción: Este operador escoge cromosomas entre la población para efectuar la reproducción. Cuanto más capaz sea el cromosoma, más veces será seleccionado para reproducirse.

- Cruce: Se trata de un operador cuya labor es elegir un lugar, y cambiar las secuencias antes y después de esa posición entre dos cromosomas, para crear nueva descendencia (por ejemplo, las cadenas 10010011 y 11111010 pueden cruzarse despus del tercer lugar para producir la descendencia 10011010 y 11110011). Imita la recombinación biológica entre dos organismos haploides [1].

[1] Una célula haploide es aquella que contiene un solo juego de cromosomas o la mitad (n, haploide) del número normal de cromosomas, en células diploides (2n, diploide). Las células reproductoras, como los óvulos y los espermatozoides de los mamíferos y algunas algas contienen un sólo juego de cromosomas, mientras que el resto de las células de un organismo superior suelen tener dos juegos de ellos.

- Mutación: Este operador produce variaciones de modo aleatorio en un cromosoma (por ejemplo, la cadena 00011100 puede mutar su segunda posición para dar lugar a la cadena 01011100). La mutación puede darse en cada posición de un bit en una cadena, con una probabilidad, normalmente muy pequeña (por ejemplo 0.001).

Programación genética

Es una especialización de los algoritmos genéticos; donde cada individuo es un programa de computadora. Es una técnica de aprendizaje automático utilizada para optimizar una población de programas de acuerdo a una función de ajuste, aptitud o *fitness* que evalúa la capacidad de cada programa para llevar a cabo la tarea en cuestión (Koza *et al.*, 1999).

La programación genética, desarrolla programas, tradicionalmente representados con estructuras de árboles. Los árboles pueden ser fácilmente evaluados de forma recursiva. Cada nodo del árbol tiene una función como operador y cada nodo terminal tiene un operando, por lo que las expresiones matemáticas son fáciles de evolucionar y evaluar (Poli & Koza, 2014).

Para desarrollar un algoritmo de programación genética se necesitan de diversas etapas: la inicialización, la selección de operadores, definición del conjunto de terminales, definición del conjunto de funciones, la selección de la función objetivo y definir la condición de paro.

Inicialización. El primer paso en programación genética consiste en formar la población inicial de individuos.

Uno de los parámetros principales para un algoritmo genético es el tamaño máximo de un programa. Este límite puede estar impuesto sobre el número de nodos o sobre la profundidad del árbol.

Selección de operadores. Generalmente se utilizan los operadores que se mencionaron en la sección anterior, es decir selección, cruce y mutación aunque pueden existir otros operadores como: permutación, expansión o duplicación del gen.

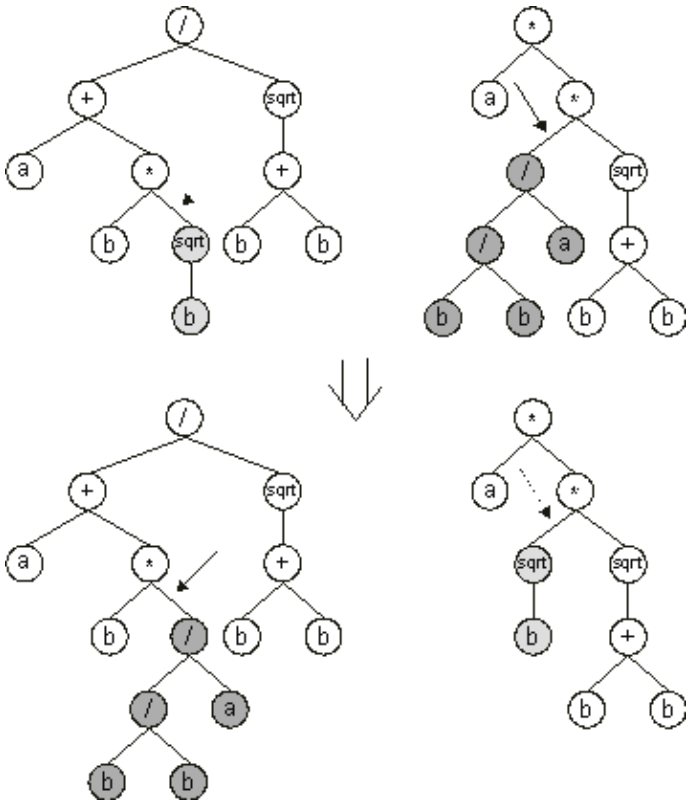

Figura 3.5: Ejemplo del operador de cruce

En la imagen 3.5 se puede observar como se lleva a cabo la operación de cruce entre dos árboles (individuos). Se puede apreciar como intercambian una de sus ramas cada uno de ellos. Por otro lado en la imagen 3.6 se puede observar un ejemplo de la operación de mutación, donde una de las ramas del individuo cambia por completo.

Definición del conjunto de terminales. Estos son los elementos que finalmente se representarán en las hojas de los árboles o individuos.

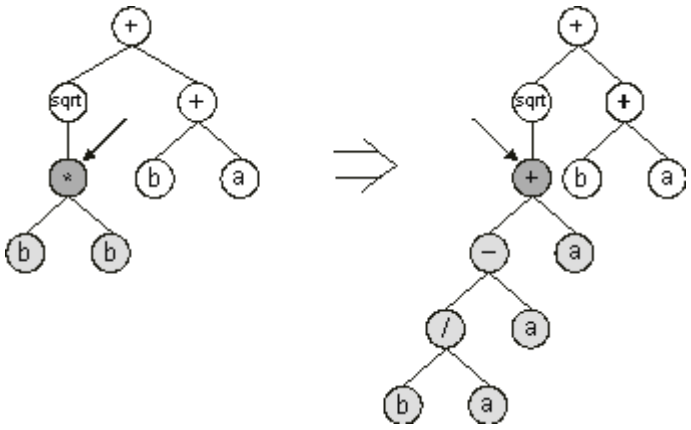

Figura 3.6: Ejemplo del operador de mutación

Definición del conjunto de funciones. Estos son los nodos internos en los árboles o individuos.

Función objetivo o de calidad. La calidad es la medida usada por el algoritmo de programación genética durante la evolución simulada para medir qué tan buena es una solución encontrada hasta el momento. Esta funcion, por lo general devuelve un valor entre cero y uno. Mucho algoritmos de programación genética intentan minimizar la función objetivo por lo que si la meta es llegar a maximizar los valores de esta función, basta con negar el resultado de la misma.

Condición de paro. Depende de lo que se desee, pero en general cuando el mejor individuo de la población tiene una calidad aceptable es cuando se puede detener el algoritmo. Aunque también se puede definir un límite de generaciones o iteraciones.

Capítulo 4

Distancias de edición enriquecidas con información semántica

Las distancias de edición han sido métodos utilizados en diversas tareas, en especial, si estas tareas consisten en encontrar similitud entre dos objetos, estructuras o por supuesto, textos.

Una distancia de edición entre dos objetos es un método que se encarga de elegir entre un conjunto de operaciones válidas, el menor número de cambios necesarios para transformar uno de los objetos en el otro. Los conjuntos de operaciones válidas dependean de cada distancia de edición aunque, las operaiones más comunes son la eliminación, inserción, sustitución y permutación de las partes del objeto que se está editando.

Existen diversas distancias de edición que se pueden elegir en dependencia de la tarea que se quiere resolver dentro de la detección de similitud textual.

Como se mencionó en el capítulo 2 una de las distancias de edición más famosas es la distancia de Levenshtein la cual ha sido objeto de muchas investigaciones desde que se diseñó en 1966 (Levenshtein, 1966).

Existen otras distancias como la Needleman-Wunsch. Esta medida originalmente se

diseñó para alinear secuencias biológicas pero, por su comportamiento es fácil aplicarla a la detección de similitud textual. Needleman-Wunsch es de naturaleza global, esto quiere decir que aplicada a detección de similitud textual, trataría de encontrar coincidencias de un texto a lo largo del segundo texto completo. Esto parecería ideal para tareas como la detección de paráfrasis ya que para generar paráfrasis la persona que lleva a cabo este proceso lo hace de forma similar a un alineamiento global, es decir, cuando un texto es paráfrasis de otro se tiene coincidenacias a lo largo de los dos textos.

Por otro lado, existe la medida de Smith-Waterman. Esta medida al igual que la anterior se propuso con la idea de alinear secuencias biológicas, pero de igual forma se puede aplicar a textos. La principal diferencia con el método de Needleman-Wunsch es que Smith-Waterman es de naturaleza local. Esto quiere decir que no trata de encontrar coincidencias de un texto en todo el largo del otro sino que trata de encontrar coincidencias locales, es decir, toma pedazos de un texto y lo trata de alinear con un pedazo del otro hasta que termine de alinear. Esto es muy parecido al proceso que se lleva a cabo cuando un texto es plagiado ya que la persona que plagia un texto, regularmente no lo hace completamente, sino que toma pedazos locales del texto original para insertarlos en el texto plagiado.

Estas distancias de edición, han dado buenos resultados desde que se diseñaron, pero las tres mencionadas comparten una desventaja. Supongamos que se desea comparar el texto "El terremoto acabó con todo a su paso" con los textos "El temblor acabó con todo a su paso" y "El perro acabó con todo a su paso". Si bien, las tres frases solo cambian en una palabra, esa palabra hace que cambia el sentido de las oraciones. Si una persona tuviera que evaluar qué tan parecidas son las frases, sin duda calificaría como identicas las frases del tembor y el terremoto mientras que la del perro le daría un grado de similitud menor. El problema es que para estas distancias de edición las frases

son igual de diferentes. Esto se debe a que las distancias de edición en general utilizan información como traslape de conjuntos de palabras y posicionamiento de las mismas pero no utilizan información semántica, lo que ocasiona que pase por alto sinónimos y palabras similares.

Por esta razón, las propuesta de este capítulo consiste en tomar estas tres distancias de edición y enriquecerlas con información semántica para poder encontrar un mejor grado de similitud textual.

4.1. Distancia de Levenshtein

4.1.1. Distancia Levenshtein original

En este método, el grado de similitud entre dos textos A y B se calcula con base en el conjunto mínimo de operaciones de edición necesarias para transformar del texto A en B, o viceversa. Como ya se mencionó, Levenshtein cuenta con tres operaciones de edición válidas: eliminación, inserción y substitución. Entre más cerca de cero es la distancia de Levenshtein más parecidos son los textos.

Formalmente, dados dos textos a y b, la distancia de Levenshtein está definida como:

Caso base:

$$Lev_{a,b}(i,0) = i$$
$$Lev_{a,b}(0,j) = j$$

Algoritmo de Levenshtein:

$$Lev_{a,b}(i,j) = min \begin{cases} Lev_{a,b}(i-1,j)+1 & \text{Eliminar} \\ Lev_{a,b}(i,j-1)+1 & \text{Insertar} \\ Lev_{a,b}(i-1,j-1)+1_{a_i \neq b_j} & \text{Sustituir} \end{cases} \tag{4.1}$$

Por ejemplo si tenemos el texto "El libro es interesante" y lo comparamos con "El libro realmente fue bueno", Levenshtein hará lo siguiente:

1. El libro es interesante → El libro realmente interesante (Sustitución de "es" por "realmente")

2. El libro realmente interesante → El libro realmente fue interesante (Inserción de "fue")

3. El libro realmente fue interesante → El libro realmente fue bueno (Sustitución de "interesante" por "bueno")

Para poder transformar el texto "El libro es interesante" al texto "El libro realmente fue bueno" Levenshtein aplicó tres movimientos (sustitución, inserción y de nuevo sustitución) por lo que el resultado de la distancia de edicón sería tres.

En el algoritmo 3 se describe la implementación de la medida de Levenshtein.

4.1.2. Modificación de la distancia de Levenshtein

El problema del algoritmo de Levenshtein es que no tiene información alguna del costo de eliminar o insertar una palabra; así como tampoco le da mucha importancia a las palabras que sustituye por otras. Para este algoritmo cada operación tiene un costo de uno y se decide si dos cadenas son cercanas dependiendo de qué tan bajo o alto sea el valor resultante. Sin embargo pueden existir casos de sinónimos o palabras semánticamente cercanas entre sí que el algoritmo pasaría por alto.

Algoritmo 3 Algoritmo de Levenshtein original

Entrada: TextoA y TextoB

Salida: Mínimo número de operaciones posibles para convertir el TextoB en el TextoA.

```
1: entero D[Len(TextoA) + 1 ][Len(TextoB) + 1], i, j, costo
2: para i = 0 hasta Len(TextoA) hacer
3:    D[i][0] := i
4: fin para
5: para i = 0 hasta Len(TextoB) hacer
6:    D[0][i] := i
7: fin para
8: para i = 1 hasta Len(TextoA) hacer
9:    para j = 1 hasta Len(TextoB) hacer
10:      si TextoA[i] = TextoB[j] entonces
11:         costo := 0
12:      si no
13:         costo := 1
14:      fin si
15:      D[i][j] := min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + costo)
16:   fin para
17: fin para
18: devolver  D[len(TextoA)][len(TextoB)]
```

Para introducir información semántica se utilizará la medida de Wu y Palmer descrita en la ecuación 2.10 en el capítulo 2 y que está disponible en *WordNet*(Wu & Palmer, 1994). Esta es una de las medidas más utilizadas y tiene la ventaja de estar acotada entre cero y uno, además de no depender de ningún corpus por lo que el resultado de la misma será determinista.

La primer modificación de este algoritmo consiste en que si se introducen o eliminan palabras vacías, estas no tendrán ningún costo. De no ser palabra vacía, en las operaciones de inserción y eliminación el costo será de uno. Para resolver este problema se eliminan todas las palabras vacías de las dos frases a comparar.

El siguiente paso a resolver sería cuando se sustituyen pares de palabras. Para esto se deben encontrar los pares de palabras más cercanas semánticamente en los dos textos, por lo que se comparan todas las palabras del texto A con todas las palabras del texto B utilizando la medida de Wu y Palmer. Cuando se encuentran los pares de palabras más cercanos se obtiene la similitud entre cada uno de ellos y ese será el costo de sustitur cada par de palabras.

Finalmente, se aplica el procedimiento normal de Levenshtein y el resultado será normalizado entre cero y uno dividiendo la suma del costo de todas las operaciones entre el máximo número entre la longitud del texto A y del texto B.

En el algortimo 4 se describe la implementación de esta modificación. En la línea 13 se utiliza la función de similitud de Wu y Palmer.

De esta manera, este algoritmo es capaz de castigar más a pares de palabras que sean lejanas entre sí y por el contrario; es posible que no castigue o que castigue en menor medida pares de palabras cercanas.

Por ejemplo, si evaluamos la similitud del par de frases "El terremoto acabó con todo a su paso" y "El temblor acabó con todo a su paso" (en inglés) con el algoritmo

Algoritmo 4 Algoritmo de Levenshtein enriquecido con información semántica

Entrada: TextoA y TextoB

Salida: Distancia de Levenshtein suavizada entre el TextoA y el TextoB.

```
1: real D[Len(TextoA) + 1 ][Len(TextoB) + 1], costo
2: diccionario Sim,Tabla
3: entero VisitadosA[Len(TextoA)], VisitadosB[Len(TextoA)], i, j, k
4: TextoA = QuitarPalabrasVacias(TextoA)
5: TextoB = QuitarPalabrasVacias(TextoB)
6: k = 0
7: para i = 0 hasta Len(TextoA) hacer
8:    VisitadosA[i] := 1
9:    para j = 1 hasta Len(TextoB) hacer
10:      VisitadosB[j] := 1
11:      Tabla[k][0] := TextoA[i]
12:      Tabla[k][1] := TextoB[j]
13:      Tabla[k][2] := WUP(TextoA[i],TextoB[j])
14:      k := k + 1
15:   fin para
16: fin para
17: OrdenarTabla(Tabla)
18: para i = 0 hasta Len(Tabla) hacer
19:   si VisitadosA[Tabla[i][0]] := 1 and VisitadosB[Tabla[i][1]] := 1 entonces
20:      VisitadosA[Tabla[i][0]] := 0
21:      VisitadosB[Tabla[i][1]] := 1
22:      Sim[Tabla[i][0]][Tabla[i][1]] = 1 - Tabla[i][2]
23:   fin si
24: fin para
25: para i = 0 hasta Len(TextoA) hacer
26:   D[i][0] := i
27: fin para
28: para i = 0 hasta Len(TextoB) hacer
29:   D[0][i] := i
30: fin para
31: para i = 1 hasta Len(TextoA) hacer
32:   para j = 1 hasta Len(TextoB) hacer
33:      si TextoA[i] = TextoB entonces
34:         costo := 0
35:      si no
36:         costo := sim[TextA[i]][TextoB[j]]
37:      fin si
38:      D[i][j] := min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + costo)
39:   fin para
40: fin para
41: devolver  D[len(TextoA)][len(TextoB)] / max(Len(TextoA), Len(TextoB))
```

original de Levenshtein, la matriz resultante quedaría como sigue:

$$lev = \begin{pmatrix} & The & earthquake & shattered & everything & in & its & path \\ The & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ quake & 1 & 1 & 1 & 2 & 3 & 4 & 5 \\ destroyed & 2 & 2 & 2 & 2 & 3 & 4 & 5 \\ everything & 3 & 3 & 3 & 2 & 3 & 4 & 5 \\ in & 4 & 4 & 4 & 4 & 2 & 3 & 4 \\ its & 5 & 5 & 5 & 5 & 5 & 2 & 3 \\ path & 6 & 6 & 6 & 6 & 6 & 3 & 2 \end{pmatrix}$$

El resultado final se obtiene de la posición de la última columna con el último renglón, es decir, el resultado de la distancia de edición es dos. Por otro lado, si se mide este mismo par de frases con la modificación propuesta, la matriz resultante quedaría de esta forma:

$$lev = \begin{pmatrix} & earthquake & shattered & everything & path \\ quake & 0 & 1 & 2 & 3 \\ destroyed & 1 & 0 & 1 & 2 \\ everything & 2 & 1 & 0 & 1 \\ path & 3 & 2 & 1 & 0 \end{pmatrix}$$

De esta forma la distancia es cero, por esta razón para la modificación de la distancia de Levenshtein, estas dos frases son idénticas.

4.2. Distancia de Needleman-Wunsch

4.2.1. Distancia de Needleman-Wunsch original

Como ya se mencionó en el capítulo 3, el algoritmo de Needleman-Wunsch es un algoritmo dinámico que sirve para realizar alineamientos globales de dos secuencias biológicas. Al alinear los caracteres de una secuencia se determina que en ambas cadenas hay caracteres que se refieren a lo mismo dentro del mismo contexto.

Esta matriz representa el costo o beneficio de alinear caracteres. Por ejemplo, en esta matriz se puede ver que es mejor alinear un par de "A" que un par de "G". Tambien se puede ver que esta matriz maneja valores negativos que sin duda son pares de caracteres que no se desea que resulten ser alineados, por ejemplo, es evidente que no se desea que se alinien los caracteres "G" y "T".

Este algoritmo construye una segunda matriz donde las filas representan cada uno de los caracteres de las secuencia uno y las columnas cada uno de los caracteres de la secuencia dos. Se recorre esta matriz y se va obteniendo el mejor valor hasta el momento entre no alinear al caracter de la secuencia uno, no alinear al caracter de la secuencia dos o alinearlos. Al no alinear también se paga un costo que se obtiene como un parámetro del algoritmo. Así, las posiciones de la segunda matriz se calculan como:

$$NW(i,j) = Max(NoAlinearCaracter(i), NoAlinearCaracter(j), AlinearCaracteres(i,j)) \tag{4.2}$$

En el algoritmo 1 se puede ver la implementación de Needleman-Wunsch (Lees *et al.*, 2014).

4.2.2. Modificación de la distancia de Needleman-Wunsch

Para que este método funcione con texto, la matriz de información debe construirse a partir de las palabras que componen los textos y no a partir de sus caracteres. En esta matriz M el elemento M_{ij} se refiere a la similitud semántica entre la i-ésima palabra del texto uno y la j-ésima palabra del texto dos.

Un primer acercamiento a la solución sería crear la matriz con unos si las palabras son identicas o cero si son distintas. Por ejemplo si desea comparar las frases "The quake destroyed everything in its path" y "The earthquake shattered everything in its path" la matriz informativa resultaría así:

$$\begin{pmatrix} & \textit{The} & \textit{earthquake} & \textit{shattered} & \textit{everything} & \textit{in} & \textit{its} & \textit{path} \\ \textit{The} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \textit{quake} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \textit{destroyed} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \textit{everything} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \textit{in} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \textit{its} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \textit{path} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

El problema de esta contrucción es que al comparar palabras semánticamente similares como *quake* y *earthquake* el método no se entera de que son palabras casi identicas en significado y al hacer el alineamiento de palabras no se llegaría a un buen resultado.

De hecho la alineación de este par de frases quedaría como:

The	*earthquake*	*shattered*	–	–	*everything*	*in*	*its*	*path*	
|					|	|	|	|
The	–	–	*quake*	*destroyed*	*everything*	*in*	*its*	*path*	

Al igual que la distancia de edición pasada, con solo información léxica el algoritmo no es capaz de alinear las palabras *quake* y *earthquake* ni las palabras *shattered* y *destroyed*. Por esta razón se utilizará la información de *WordnNet* para que pueda ser incluida en la tabla y obtener un buen alineamiento final de palabras.

En el algoritmo 5 se muestra cómo se calcula la tabla informativa.

Algoritmo 5 CalculoDeLaTablaS

Entrada: TextoA, TextoB
Salida: La tabla informativa S

```
1: entero i,j
2: para i = 1 hasta Len(TextoA) hacer
3:    para j = 1 hasta Len(TextoB) hacer
4:       S[i][j] := WUP(TextoA[i], TextoB[j])
5:    fin para
6: fin para
7: devolver RecalculoDelaTabla(S)
```

Donde el procedimiento de RecalculoDeLaTabla se define en el algoritmo 6. Este procedimiento se lleva a cabo para normalizar los valores entre -1 y 1 y para alejar los valores de las palabras que más se parecen a los que menos se parecen.

De esta manera la nueva matriz informativa, para el ejemplo antes mencionado quedaría de la siguiente forma:

Algoritmo 6 RecaalculoDeLaTabla

Entrada: Una Tabla S
Salida: La tabla informativa S recalculada

```
1: entero matyor, segundomayor, i,j
2: para i = 1 hasta Len(S) hacer
3:    mayor = ObtenerElementoMayordelRenglon(i,S)
4:    segundoMayor = ObtenerSegundoElementoMayordelRenglon(i,S)
5:    para j = 1 hasta Len(S[i]) hacer
6:       si S[i][j] ¡mayor entonces
7:          S[i][j] = S[i][j] - mayor + segundomayor
8:       fin si
9:    fin para
10: fin para
11: devolver S
```

	The	*earthquake*	*shattered*	*everything*	*in*	*its*	*path*
The	1	−1	−1	−1	−1	−1	−1
quake	−1	1	−1	−1	−1	−1	−1
destroyed	−1	−1	1	−1	−1	−1	−1
everything	−1	−1	−1	1	−1	−1	−1
in	−1	−1	−1	−1	1	−1	−1
its	−1	−1	−1	−1	−1	1	−1
path	−1	−1	−1	−1	−1	−1	1

De esta forma, la matriz informativa contiene la relación semántica de las palabras para poder hacer alineaciones de mejor forma. El procedimiento no castiga cuando no se alinean palabras vacías, mientras que si no se alinean palabras no vacías, entonces eso se castiga con uno.

Finalmente, el algoritmo modificado de Needleman-Wunsch retorna el valor de la mejor alineación dividido entre el menor número entre la longitud del texto A y B como

se muestra en el algoritmo 7. En el ejemplo anterior esta alineación quedaría como:

The	*earthquake*	*shattered*	*everything*	*in*	*its*	*path*
|	|	|	|	|	|	|
The	*quake*	*destroyed*	*everything*	*in*	*its*	*path*

Es decir, se alcanzó una alineación perfecta.

4.3. Distancia de Smith-Waterman

4.3.1. Distancia de Smith-Waterman original

Este algoritmo, al igual que el de Needleman-Wunsch, es para alinear secuencias, con la diferencia de que este algoritmo trata de encontrar el mejor segmento local de uno de los textos en el otro. Al igual que el procedimiento anterior, las operaciones de edición permitidas son: No alinear caracter de la secuencia uno, no alinear caracter de la secuencia dos, alinear caracteres, aunque una de las diferencias es que esta permite una operación más la cual consiste en terminar uno de los alineamientos locales. En el algoritmo 2 se muestra el procedimiento completo de Smith-Waterman (Liu & Schmidt, 2014).

4.3.2. Modificación de la distancia de Smith-Waterman

Si se hace el mismo procedimiento de alinear las frases "The quake destroyed everything in its path" y "The earthquake shattered everything in its path" con una matriz binaria el resultado de la alineación sería el siguiemnte:

Algoritmo 7 Algoritmo de Needleman-Wunsch con información semántica

Entrada: TextoA, TextoB

Salida: El costo de la mejor alineación global entre los TextoB y TextoA.

```
1: real D[Len(TextoA) + 1 ][Len(TextoB) + 1], i, j, Alinear, DejarHuecoEnA, Dejar-
   HuecoEnB, S
2: entero d, cont
3: S := CalculoDeLaTablaS(TextoA, TextoB)
4: cont := 0
5: para i = 0 hasta Len(TextoA) hacer
6:    d := 0
7:    si i > 1 and NoEsPalabraVacia(TextoA[i]) entonces
8:       d := -1
9:    fin si
10:   cont := cont + d
11:   D[i][0] := cont
12: fin para
13: cont := 0
14: para i = 0 hasta Len(TextoB) hacer
15:   d := 0
16:   si i > 0 and NoEsPalabraVacia(TextoB[i]) entonces
17:      d := -1
18:   fin si
19:   cont := cont + d
20:   D[i][0] := cont
21: fin para
22: para i = 1 hasta Len(TextoA) hacer
23:   para j = 1 hasta Len(TextoB) hacer
24:      Alinear := D[i-1][j-1] + S[TextoA[i]][TextoB[j]]
25:      d := 0
26:      si NoEsPalabraVacia(TextoA[i]) entonces
27:         d := -1
28:      fin si
29:      DejarHuecoEnA := D[i-1][j] + d
30:      si NoEsPalabraVacia(TextoB[j]) entonces
31:         d := -1
32:      fin si
33:      DejarHuecoEnB := D[i][j-1] + d
34:      D[i][j] := max(Alinear, DejarHuecoEnA, DejarHuecoEnB)
35:   fin para
36: fin para
37: devolver  D[len(TextoA)][len(TextoB)] / min(len(TextoA), len(TextoB))
```

The	*earthquake*	–	*shattered*	–	*everything*	*in*	*its*	*path*	
|					|	|	|	|
The	–	*quake*	–	*destroyed*	*everything*	*in*	*its*	*path*	

De nueva cuenta, un algoritmo de alineamiento de secuencias con solo informacón léxica fue incapaz de hacer un alineamiento eficaz de dos frases. Para poder llegar a un mejor alineamiento, en ese caso local, de dos frases se utilizará la misma tabla informativa calculada en el algoritmo anterior. Al igual que en las dos modificaciones anteriores las inserciones y eleminaciones de palabras vacias no se castigan.

En el algoritmo 8 se muestra la modificación completa

Con esta modificación, las frases de ejemplo quedarían alineadas así:

The	*earthquake*	*shattered*	*everything*	*in*	*its*	*path*
|	|	|	|	|	|	|
The	*quake*	*destroyed*	*everything*	*in*	*its*	*path*

De nueva cuenta, se llega a una alineación perfecta.

4.4. Conclusiones del capítulo

En este capítulo se presentaron modificaciones de los algoritmos de Levenshtein, Needleman-Wunsch y Smith-Waterman. Siendo el primero la distancia de edición más famosa hasta el día de hoy, mientras que los otros dos algoritmo fueron creados para hacer alineación de secuencias biológicas.

Se espera que estas modificaciones sean capaces de encontrar un mejor grado de

Algoritmo 8 Algoritmo de Smith-Waterman con información semántica

Entrada: TextoA, TextoB

Salida: El costo de la mejor alineación local entre los TextoB y TextoA.

```
1: entero D[Len(TextoA) + 1 ][Len(TextoB) + 1], i, j, Alinear, DejarHuecoEnA, Dejar-
   HuecoEnB, costo, d, S
2: para i = 0 hasta Len(TextoA) hacer
3:    D[i][0] := 0
4: fin para
5: para i = 0 hasta Len(TextoB) hacer
6:    D[0][i] := 0
7: fin para
8: S := CalculoDeLaTablaS(TextoA, TextoB)
9: para i = 1 hasta Len(TextoA) hacer
10:   para j = 1 hasta Len(TextoB) hacer
11:      Alinear := D[i-1][j-1] + S[TextoA[i]][TextoB[j]]
12:      d := 0
13:      si NoEsPalabraVacia(TextoA[i]) entonces
14:         d := -1
15:      fin si
16:      DejarHuecoEnA := D[i-1][j] + d
17:      si NoEsPalabraVacia(TextoB[j]) entonces
18:         d := -1
19:      fin si
20:      DejarHuecoEnB := D[i][j-1] + d
21:      D[i][j] := max(Alinear, DejarHuecoEnA, DejarHuecoEnB)
22:   fin para
23: fin para
24: devolver D
```

similitud textual que los algoritmos originales. Estas tres modificaciones serán evaluadas y los resultados serán reportados más adelante en el capítulo 6.

Capítulo 5

Generación automática de funciones de similitud textual

DISTANCIAS DE EDICIÓN CON INFORMACIÓN SEMÁNTICA

En el capítulo 4 se propusieron tres maneras distintas de calcular el grado de similitud semántica sobre textos cortos de forma no supervisada. En este capítulo se presenta un método para generar de forma automática y a través de un algoritmo de programación genética, funciones para resolver la misma tarea; la detección de similitud textual.

Esta propuesta consiste básicamente en combinar algunas medidas de detección de similitud ya existentes como las que se mencionaron en el capítulo 2 y otras definidas más adelante en este capítulo, con la intención de encontrar una buena relación entre estas técnicas y la tarea que queremos resolver.

En la sección 3.3 se mencionó de manera breve que son los algoritmos evolutivos y en la sección 3.3.2 se abordó particularmente el tema de programación genética. A través de esta técnica se propone mezclar tres grupos de medidas para detectar similitud; con la intención de obtener a partir de un grupo de resultados un resultado final que sea mejor que cada medida individual. Estos grupos son los siguientes:

- Medidas que consideran el traslape de conjuntos de palabras
- Medidas que consideran la secuencia de las palabras
- Medidas que consideran sustituciones o transformaciones semánticas

En las siguientes secciones se explica en qué consiste cada grupo, primero se describiran las medidas existentes que se seleccionaron y posteriormente. las medidas propuestas en este trabajo.

5.1. Medidas base consideradas

5.1.1. Medidas que consideran el traslape de conjuntos de palabras

En este grupo, los textos serán tratados como conjuntos de palabras y serán procesados por algunas de las medidas descritas en el capítulo 2. Las medidas seleccionadas para este grupo son:

- Coeficiente de Dice (ecuación 2.14)
- Coeficiente de Jaccard (ecuación 2.15)
- Coeficiente de traslape (ecuación 2.17)
- Coeficiente de coseno (ecuación 2.18)

5.1.2. Medidas que consideran la secuencia de las palabras

En este grupo, se utilizarán algoritmos que detectan similitud en las secuencias de las palabras de los textos que se van a comparar tomando en cuenta el posicionamiento

absoluto y relativo de las palabras en cada frase. Las medidas que se utilizarán en esta categoría son (igualmente la descripción de estos algoritmos se encuentra en el capítulo 2):

- Algoritmo de la subsecuencia común más larga
- Algoritmo de la subsecuencia común más larga sobre palabras vacías
- Algoritmo de Levenshtein
- Algoritmo de Levenshtein sobre palabras vacías
- Coeficiente de Dice sobre bi-gramas
- Coeficiente de Jaccard sobre bi-gramas
- Coeficiente de traslape de conjuntos sobre bi-gramas
- Coeficiente de coseno sobre bi-gramas

5.1.3. Medidas que consideran sustituciones o transformaciones semánticas consideradas

En esta categoría se utilizarán los tres métodos descritos en el capítulo anterior, aparte de algunos métodos que se propondrán más adelante; por lo pronto los métodos en esta categoría son:

- Distancia de Levenshtein con información semántica
- Algoritmo de Neddleman-Wunsch con información semántica
- Algoritmo de Smith-Waterman con información semántica

5.2. Otras medidas propuestas

5.2.1. Posicionamiento de palabras

Aparte de estas ocho formas que ya se eligieron para detectar similitud a partir del posicionamiento de las palabras se propone un método más para detectar la similitud a partir de las posiciones relativas de las palabras compartidas entre ambos textos.

Este algoritmo encuentra las palabras que comparten ambos textos y toma en cuenta su posición en las frases y también las posiciones que ocupan las palabras que están a su alrededor. El procedimiento suma la diferencia entre las posiciones y finalmente retorna este resultado.

En el algoritmo 9 se describe formalmente el procedimiento.

Algoritmo 9 Algoritmo de similitud de posicionamiento de palabras

Entrada: TextoA y TextoB
Salida: Suma de las diferencias relativas de la palabra compartidas en ambos textos normalizado entre cero y uno

```
1: entero i, j, k, suma
2: diccionario tabla
3: k = 0
4: para i = 1 hasta Len(TextoA) hacer
5:    para j = 1 hasta Len(TextoB) hacer
6:       si TextoA[i] = TextoB[j] entonces
7:          tabla[k] = j
8:          k = k + 1
9:          Terminar ciclo
10:      fin si
11:   fin para
12: fin para
13: suma = 0
14: para i = 1 hasta Len(tabla) - 1 hacer
15:    suma = suma + |tabla[i] - tabla[i + 1]| - 1
16: fin para
17: devolver  suma / len(tabla)
```

5.2.2. Medidas que consideran sustituciones o transformaciones semánticas

En este grupo se consideran medidas que toman en cuenta la cercanía semántica de los textos. Medidas como el coeficiente de Dice, no toman en cuenta que dos palabras son similares o no, solo saben si son estrictamente iguales. El problema es que existen diversas formas de escribir una misma idea; esto hace posible que dos textos sean similares a pesar de compartir pocas palabras estrictamente iguales. Por esta razón se proponen algunas extensiones al coeficiente de Dice, de tal forma que puedan capturar sustituciones o transformaciones semánticas.

Coeficiente de Dice con intersección semántica

Para esta medida se intentará capturar la cercanía semántica que exista entre las palabras que no se encuentren en la intersección tradicional. Esto con la finalidad de que los sinónimos y palabras que pertenezcan a un mismo campo semántico afecten en la menor medida al grado de similitud.

Para medir la cercanía semántica de las palabras se utilizará la función WUP de similitud de palabras incluida en *WordNet*. Esta extensión está definida como:

$$CD_{IS} = 2\frac{|A \cap B| + SimWordNet(A - (A \cap B), B - (A \cap B))}{|A| + |B|} \tag{5.1}$$

Donde A y B son los conjuntos de palabras de las frases que se analizan. La función *SimWordNet(X, Y)* se encarga de comparar todas las palabras del conjunto X con las palabras del conjunto Y mediante *WordNet*.

La función *SimWordNet(X, Y)* está definida como:

$$SimWordNet(X,Y) = \sum_{w_i \in X} \sum_{w_j \in Y} WUP(w_i, w_j) - Overlap(w_i, w_j) \tag{5.2}$$

Donde:

$$Overlap(x,y) = \begin{cases} 0 & \text{si no se han contabilizado ni x ni y} \\ WUP(x,y) & \text{en otro caso} \end{cases} \tag{5.3}$$

Coeficiente de Dice con intersección semántica clusterizada

La intención de esta medida es, al igual que la medida anterior, capturar además de las palabras idénticas, las palabras semánticamente cercanas entre sí que no sean estrictamente iguales. Un inconveniente que podría tener la medida anterior es que no cuenta con un umbral para dejar de sumar los valores de similitud de cada par de palabras. El problema es que existen valores tan pequeños que no dan evidencia de alguna similitud. Por esta razón se necesita establecer un límite para saber hasta dónde dejar de contabilizar estos valores.

Para tratar de solucionar este problema, independientemente de la medida que se utilice para obtener las similitudes de palabras, se propone agrupar los valores de similitud que se obtengan de la comparación de todas las palabras del primer texto con las palabras del segundo. Para esto se propone utilizar el algoritmo *k-means* para separar los datos en grupos, donde los pares de palabras con mayor grado de similitud se agruparán en un cluster mientras que las de menor similitud se agruparán en los demas. Así, sólo se sumarán lo valores que se encuentren en el cluster con las similitudes más altas respetando el hecho de que no se puedan repetir palabras.

Elección del número de clusters Uno de los parámetros que necesita el algoritmo *k-means* es el número de clusters o mejor conocido como k. Es difícil establecer un valor que funcione bien para todos los pares de de textos que se van a evaluar. Afortunadamente, ya hay trabajos que recomiendan algunas funciones para calcular el valor de k de forma automática. Para este trabajo, se propone utilizar la técnica Dougherty (Dougherty *et al.*, 1995; Álvarez *et al.*, 2013), la cual define a k como $k = max(1, log_2(n))$ donde n es el número de elementos que se van a agrupar.

Esta extensión se define como:

$$CD_{ISC} = 2\frac{|A \cap B| + SimWordNetC(A - (A \cap B), B - (A \cap B))}{|A| + |B|} \tag{5.4}$$

Donde:

$$SimWordNetC(X, Y) = \sum_{w_i \in X} \sum_{w_j \in Y} WUP(w_i, w_j) - OverlapC(w_i, w_j) \tag{5.5}$$

Donde:

$$OverlapC(x, y) = \begin{cases} 0 & \text{si no se han contabilizado ni x ni y} \\ & \text{y si se ubica en el cluster con valores mas altos} \\ WUP(x, y) & \text{en otro caso} \end{cases} \tag{5.6}$$

Coeficiente de Dice por generalidad basado en conocimiento

Esta extensión se basa en la idea de que mientras más general sea una palabra compartida en las dos frases menos valor o peso se le debe de atribuir. Esto es porque no

debería importar lo mismo que un par de frases compartan un artículo que un verbo. Así como, si dos textos comparten palabras muy generales no debería ser indicio de que los textos son similares, sin embargo, dos textos entre más palabras particulares compartan mayor evidencia se tiene para determinar la similitud entre ellos. Por ejemplo, si dos textos hablan de medicina no necesariamente se deben considerar similares si comparten palabras como "enfermedades", "medico" o "consultorio", pero si comparten palabras como "ectrodactilia"[1], "oftalmólogo" o "estetoscopio" es más probable que los textos sean similares.

Por esta razón, esta medida trata de capturar la generalidad de cada palabra. La generalidad de cada palabra se medirá mediante la profundidad de una palabra en el árbol de *WordNet*. En la imagen 5.1 se puede ver un subconjunto de dicho árbol. La raíz es *Entity* por lo que se debe calcular la similitud de la raíz con cada palabra a evaluar.

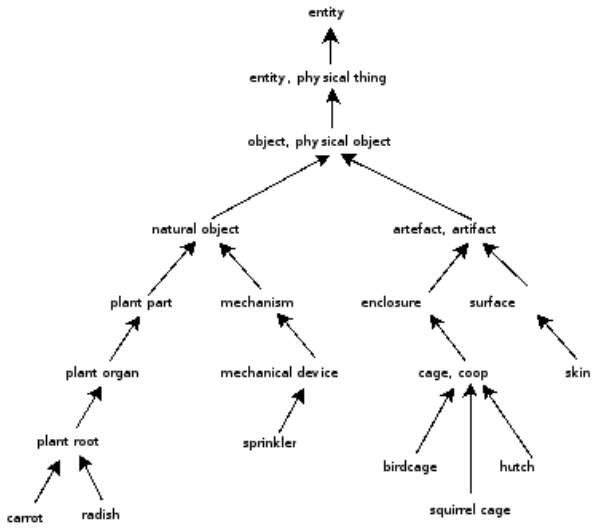

Figura 5.1: Subconjunto del árbol de *WordNet*

Esta extensión esta definida como:

[1]Es una enfermedad hereditaria con un patrón de herencia de tipo autosómico dominante. Ocasiona una malformación de las extremidades en la que hay ausencia de partes o de dedos completos

$$CD_G = 2\frac{\sum_{w \in A \cap B} generality(w)}{\sum_{w_A \in A} generality(w_A) + \sum_{w_B \in B} generality(w_B)} \tag{5.7}$$

Donde la función $generality()$ se calcula como:

$$generality(w) = 1 - WUP(Entity, w) \tag{5.8}$$

Coeficiente de Dice por generalidad basado en corpus

En esta propuesta se sigue la misma idea que el método anterior, pero en lugar de tomar la información general desde *WordNet* se tomará de un corpus con los valores de $tf \cdot idf$ de cada palabra, por lo que la función se define como:

$$CD_{GC} = 2\frac{\sum_{w \in A \cap B} generality_c(w)}{\sum_{w_A \in A} generality_c(w_A) + \sum_{w_B \in B} generality_c(w_B)} \tag{5.9}$$

Donde la función $generality_c()$ se calcula como:

$$generality_c(w) = tf \cdot idf(w) \tag{5.10}$$

5.3. Combinación de las medidas

En resumen, se cuenta con 20 medidas para detectar similitud entre un par de textos. Estas medidas están dividias en tres categorías. La primer categoría considera el traslape de conjuntos de palabras y las medidas elegidas son: el coeficiente de Dice, el coeficiente de Jaccard, el coeficiente de traslape y el coeficiente de coseno. Por otro lado, en la categoría de secuencia de palabras se tienen las medidas: de la subsecuencia comúm más larga, de la subsecuencia comúm más larga sobre palabras vacías, de Levenshtein, de Levenshtein sobre palabras vacías, el coeficiente de Dice sobre bi-gramas, el coeficiente

de Jaccard sobre bi-gramas, el coeficiente de traslape de conjuntos de bi-gramas, el coeficiente de coseno sobre bi-gramas y la medida de posicionamiento de palabras. Por último, las medidas de la categoría de detección de transformaciones semánticas son: los métodos descritos en el capítulo 4, el coeficiente de Dice con intersección semántica, el coeficiente de Dice con intersección semántica clusterizada, el coeficiente de Dice por generalidad basada en conocimiento, el coeficiente de Dice por generalidad basada en corpus.

La idea es a partir de los resultados de estas medidas llegar a un resultado general que mejore la detección de similitud de cada una de ellas. Mediante programación genética se propone combinar estas medidas para generar árboles donde las hojas serán las funciones antes descritas y los nodos que no sean hojas serán operaciones matemáticas que en conjunto definirán nuestra nueva función generada a través de un algoritmo de programación genética.

5.3.1. Combinación mediante programación genética

Utilizando como herramienta, programación genética se combinarán todas las medidas antes mencionadas. En la imagen 5.2 se muestra el flujo general del método propuesto. Como se puede observar, el método necesita de una fase de entrenamiento donde a partir de las medidas utilizadas y con alguna función de aptitud (*fitness*), el método intentará encontrar una buena combinación de medidas para poder ser utilizadas posteriormente en la fase de prueba.

A continuación se explicará con más detalle los módulos de programación genética y de la función de aptitud utilizada (*fitness*).

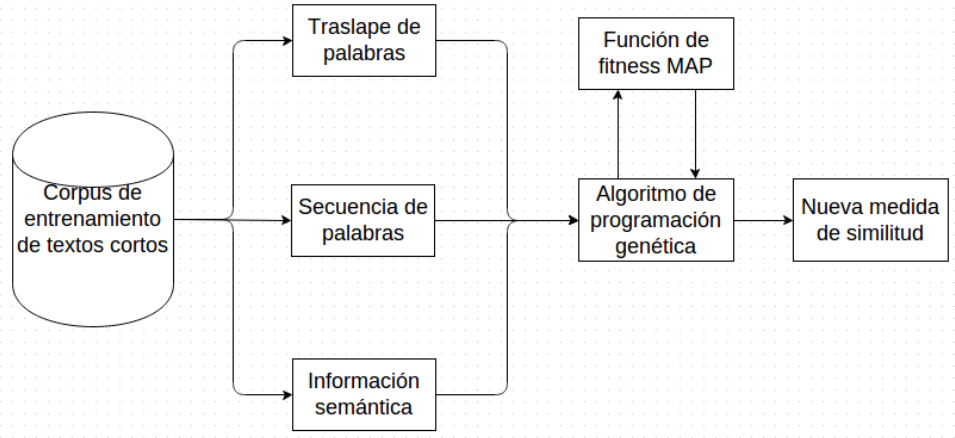

Figura 5.2: Estructura general del método para obtener funciones automáticas

Programación genética

Como se mencionó en el capítulo 3, para que un algoritmo de programación genética funcione se necesitan definir algunos parámetros iniciales como el nivel máximo de nuestros árboles, operadores, conjunto de terminales, conjunto de funciones y la función de *fitness*. A continuación se describirá cada uno de ellos con excepción de la función de *fitness* la cual se describirá en una sección diferente.

Para generar la población inicial se utilizará una técnica llamada *ramped half-and-half* (Bhardwaj *et al.*, 2014). Esta técnica consiste en inicializar a la mitad de la población mediante un método llamado *The full method*. Este método genera individuos (árboles) de manera aleatoria con la máxima profundidad definida y balanceados. Por otro lado, la segunda mitad de la población se genera con una técnica llamada *the grow method*; ésta consiste en generar, al igual que la técnica anterior, de forma aleatoria a los individuos pero no necesariamente se generan de la máxima profundidad y balanceados.

Para el número de individuos y generaciones se utilizarán los parámetros por *default* que se encuentran en la librería de *GPLAB* de matlab, estos parámetros están definidos en 50 individuos y 50 generaciones.

Uno de los principales problemas que se pueden encontrar es que la profundidad de los árboles podría crecer muy rápido por lo que se necesita definir un nivel máximo para evitar individuos que no sean legibles. Niveles pequeños como dos o tres podrían cortar el espacio de búsqueda dejando escapar posiblemente las mejores soluciones. Por otro lado números más grandes como 10 o tal vez 11 harían el espacio de búsqueda muy grande lo que haría el algoritmo muy tardado puesto que los árboles con esas profundidades son demasiado grandes tanto para ser encontrados como para ser interpretados. Un número que parece ser un buen balance entre profundidad y rapidez es el número cinco.

Los operadores serán los mismos que se describen en el capítulo 3. Estos operadores son: selección, cruce y mutación.

El conjunto de terminales serán las 20 funciones definidas en las secciones anteriores de este mismo capítulo. Por su parte, el conjunto de funciones serán operaciones matemáticas básicas como suma, resta, multiplicación, división, raíz cuadrada, potenciación y valor absoluto.

Definición de la función *fitness*

Como se mencionó en el capítulo 3, durante la evaluación de la programación genética sobre la población, se halla la solución del problema a partir de parámetros definidos con anterioridad, y se le da una puntuación a esa solución en función de lo cerca que esté de la mejor solución. A esta puntuación se le llama *fitness* (Liu *et al.*, 2014a).

Idealmente, se pensaría que en una función de similitud, entre mayor sea el valor resultante, los textos comparados son más parecidos; y entre más parecidos sean los textos, existe mayor probabilidad de que uno sea paráfrasis del otro o sea plagiado o un resumen, etc. Por esta razón se esperaría que los pares de textos que sean relevantes (muy similares) obtuvieran un grado de similitud alto, mientras que los que no son relevantes

deberían tener, por el contrario, un valor bajo.

Por este motivo, una buena función de *fitness* es la función MAP() (*Mean Average Precision*). Esta función evalua qué tan alto es el valor que obtuvieron los pares de texto que son relevantes. Si se genera una tabla donde se guarden los valores de relevancia y los del resultado de alguna función de similitud, entonces, la función MAP() dará como calificación un cero si es que los pares de textos relevantes, obtienen valores bajos. El caso ideal es cuando todos los pares de texto que son relevantes obtienen un valor de similitud más alto que todos los pares que no son relevantes; en este caso, la calificación sería uno (Aly *et al.*, 2014).

Esta función aplicada a esta tarea se define como:

$$AP(n) = \frac{\sum_{k=1}^{n}(clase(k))}{n} \tag{5.11}$$

Donde n es el número de pares de texto evaluados hasta el momento y la función $clase(k)$ devuelve uno si el $k-$ésimo par de textos pertenece a la clase positiva y cero de lo contrario.

Finalmente la función MAP se define como:

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q} \tag{5.12}$$

Para esto, los resultados obtenidos en la etapa de entrenamiento se ordenarán por el valor de similitud que se le haya otorgado a cada par de textos en el corpus. Lo que se espera es que todas las clases positivas se encuentren en la parte de arriba al ser ordenadas como se muestra en la figura 5.3. En esta imagen se puede ver que los valores más altos son los que tienen clase positiva (uno) mientras que los valores bajo son los que tienen clase negativa (cero). Este comportamiento es el que se espera que se mantenga

con la función aprendida para detectar similitud.

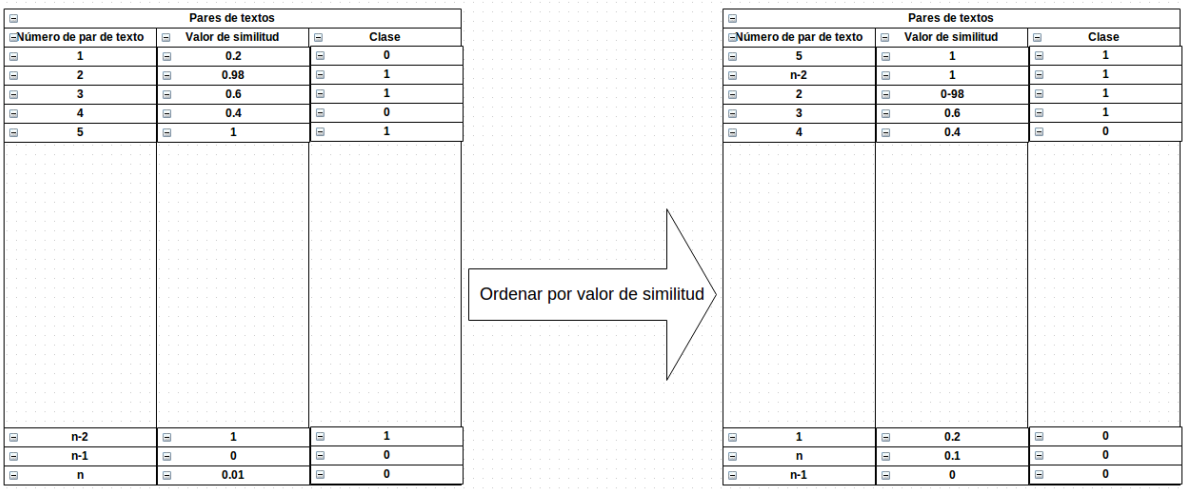

Figura 5.3: Ejemplo de los valores resultantes antes y después de ser ordenados.

Finalmente, en la tabla 5.1 se hace un resumen de los parámetros iniciales del algoritmo de programación genética.

Tabla 5.1: Definiciones iniciales del algoritmo de programación genética

Parámetro	Definido como
Población inicial	*ramped half-and-half*
Número de individuos	50
Número de generaciones	50
Nivel máximo de los individuos	5
Operadores	Selección, cruce y mutación
Terminales	Las 20 medidas descritas
Funciones	$+$,-,$*$,/,$\sqrt{}$,n^2,n^3,n^m,$\lvert n \rvert$
Función de *fitness*	Función MAP

Capítulo 6

Experimentos y resultados

En este capítulo se describe y muestra los experimentos realizados para validar los métodos propuestos en esta tesis.

Como se ha mencionado a lo largo de este texto, existen diversas tareas relacionadas con la tarea de similitud textual. Algunas tareas por sus características son importantes para evaluar los métodos propuestos en esta tesis.

En el capítulo 4 se propone un método de alineación global. Esta forma de alineación es semejante al proceso que se lleva a cabo en el fenomeno de la paráfrasis, donde se espera que un texto que sea paráfrasis de otro sea similar de forma global a lo largo del texto. Por este motivo, en este capítulo se experimentará con la tarea de detección de paráfrasis.

Por otro lado, en el mismo capítulo 4 se propone un método de alineación de naturaleza local, que de forma análoga a la tarea anterior, esta técnica se apega más a la tarea de detección de plagio, esto porque generalmente, cuando se lleva a cabo plagio en texto solo se toman pequeños pedazos del texto original para luego incluirlos en el texto que se considera sospechoso.

A su vez, otro objetivo de experimentar con dos tareas distintas (detección de paráfra-

sis y detección de plagio) es el de comparar el funcionamiento del método de generación automática de funciones propuesto en el capítulo 5 para observar el rendimiento de un mismo método en diferentes tareas.

Estas tareas son de tipo binario, es decir, la respuesta esperada es una etiqueta con el valor de sí es paráfrasis o no es paráfrasis (el mismo caso con la tarea de detección de plagio). El problema es que los métodos propuestos en este trabajo son de tipo continuo por lo que se describirá la adecuación para que los métodos sean capaces de clasificar a partir de un resultado continuo.

También se describirán las colecciones de datos con las que se experimentará. Se detallará cómo se evaluarán los métodos y cómo se compararán.

Se explicarán los *baselines* con los que se compararán cada uno de los métodos, primero describiendo el *baseline* de los métodos de distancias de edición y alineación de secuencias biológicas enriquecidas con información semántica. Después se describirá el *baseline* con el que se comparará el generador automático de funciones. Finalmente se reportarán los resultados obtenidos de cada uno de los métodos.

6.1. Colecciones de datos

6.1.1. *Microsoft Research Paraphrase Corpus*

También conocido como MSRP, este corpus está formado por 5800 pares de frases extraídas de fuentes de noticias en la web. Estos pares de frases fueron etiquetados de forma manual por expertos con dos etiquetas posibles: paráfrasis y no paráfrasis. En Dolan *et al.* (2004)[1] se puede encontrar la descripción de este corpus.

De los 8500 pares de oraciones, 4076 pares son para la fase de entrenamiento. De

[1]http://research.microsoft.com/en-us/downloads/607D14D9-20CD-47E3-85BC-A2F65CD28042/default.aspx

estos, 2753 son etiquetas positivas (es decir, que se consideran paráfrasis). Esto es el 67.5 % del corpus de entrenamiento.

Para la fase de prueba, se cuenta con 1725 pares de oraciones, siendo 1147 casos positivos, es decir, el 66.5 % de los pares totales para esta fase (Achananuparp *et al.*, 2008).

Este corpus es el mismo corpus probado en los trabajos de Mihalcea *et al.* (2006), Fernando & Stevenson (2008) y Madnani *et al.* (2012).

Un ejemplo de los pares de frases con las que cuenta este corpus es:

Frase 1: *Amrozi accused his brother, whom he called "the witness", of deliberately distorting his evidence.*

Frase 2: *Referring to him as only "the witness", Amrozi accused his brother of deliberately distorting his evidence.*

Este ejemplo está clasificado como positivo, es decir, la frase dos es considerada paráfrasis de la primera. Este ejemplo obtiene buenos resultados a partir de traslape de conjuntos de palabras ya que no existen muchos cambios semánticos de una frase a otra.

Otro ejemplo sería el siguiente:

Frase 1: *Braker said Wednesday that police, as part of protocol, were talking with other children Cruz had access to.*

Frase 2: *He told Today that authorities, as part of protocol, were talking with other children Cruz had had access to.*

En este ejemplo, a pesar de que las frases comparten varias palabras idénticas existen pequeñas modificaciones que solo con medidas que calculan traslape y secuencia de palabras no sería suficiente para determinar que una frase es paráfrasis de otra. Por ejemplo, cuando en la frase uno se hace referencia a la palabra *police* en la frase dos esta palabra se cambia por *authorities* lo cual podría ser capturado con información semántica. Otro

ejemplo es el cambio de día, mientras en la frase uno se menciona *Wednesday* en la frase dos se dice *Today*, afortunadamente, estas palabras están fuertemente relacionadas en *WordNet* lo que permite que métodos como los que se han propuesto en este trabajo capturen fenomenos como éste.

6.1.2. Corpus METER: *Measuring text reuse*

El corpus METER fue creado para medir la reutilización del texto de los cables en la redacción de los periódicos. Esta reutilización no conforma propiamente un plagio, pues los departamentos de redacción de los periódicos pagan cuotas a las agencias de noticias para poder utilizar libremente todos estos materiales. Si los periódicos no tuvieran permiso de utilizar los contenidos de los cables muchas noticias serían plagios de los cables.

El corpus METER está compuesto por 1717 documentos que se encuentran organizados en 2 grandes grupos: 944 son noticias publicadas[2] y 773 son cables de noticias de la agencia de noticias PA[3]. Está organizado de modo que cada noticia tiene su conjunto de cables, los cables y la noticia de estos subconjuntos abordan exactamente el mismo hecho. Cada noticia fue analizada y etiquetada manualmente por un reportero experto que analizó si la noticia contenía material que indicaba la utilización de alguno de los cables. Las categorías con las que se etiquetó el corpus son: no derivado, parcialmente derivado y totalmente derivado. No derivado implica que la noticia fue escrita de forma independiente a los cables; Parcialmente derivado son las noticias que utilizan alguna información de los cables, pero también existe información que es independiente. Por último, las noticias que utilizan sólo la información de los cables sin ningún material

[2]Para la recolección de noticias fueron monitoreados las secciones de leyes y de negocios de 7 periódicos británicos.

[3]Press Association, UK.

original se encuentran en la categoría de Totalmente derivado". (Gaizauskas *et al.*, 2001).

Para la evaluación de este corpus se consideró a las noticias como los documentos sospechosos y a los cables como las fuentes; hemos utilizado únicamente un subconjunto del corpus METER. Esta selección está constituida por todas las noticias que cuentan exclusivamente con un cable (es decir, una sola fuente); la selección fue necesaria debido a que las noticias que cuentan con más de un cable no indican de cuál de los cables es que la noticia fue derivada.

De esta manera, el subconjunto donde se probaron los métodos propuestos y los métodos de referencia, está constituido por 253 pares de documentos sospechoso-fuente; se consideraron los documentos con las etiquetas Parcialmente derivado" y Totalmente derivado como los casos de plagio (clase positiva) y a los No derivado como no plagiados (clase negativa). Este es el mismo corpus probado en los trabajos de Chan & Ng (2008) y Sánchez-Vega *et al.* (2013).

Un ejemplo de las frases de este corpus es:

Frase 1: *Teenage lovers left forgive us suicide note two teenage sweethearts who could not bear to be apart killed themselves after leaving a note saying please forgive us. An inquest heard today. Leanne elveck.*

Frase 2: *Teenage lovers in death pact twoteenage sweethearts who could not bear to be apart killed themselves . leaving a note begging .please forgive us . leanneelveck . 16 . and car valet damien kilburn.*

Aunque los textos están estructurados de manera un tanto complicada de leer, esta es la estructura de un cable original, aunque con un preprocesamiento adecuado esto no será problema cuando se analicen. Al igual que en los ejemplos del corpus anterior, los textos comparten palabras, secuencias de palabras y fenomenos semánticos como en las palabras *suicide* y *death*. Así los métodos propuestos en este trabajo pueden capturar

modificaciones de este estilo y pueden tomar la decisión de clasificar a un par de frases como éstas como plagio o no.

6.2. Configuración experimental

6.2.1. Pre-procesamiento de los datos

Antes de llevar a cabo algún experimento es necesario que los datos estén en un formato que sea fácil de manipular para evitar posibles errores. Puesto que los métodos propuestos solo trabajan sobre palabras es necesario remover todo lo que no sea palabras, es decir, signos de puntuación, espacios, números, etc. Luego, para poder comparar estas palabras sin problemas, todas las letras se convertirán a letras minúsculas si es que no lo están ya de esta forma. Por último, para incrementar la posibilidad de encontrar coincidencias entre palabras se lematizarán cada una de ellas para quedarse con sus respectivas raices. Para todas estas operaciones se utilizará la librería NLTK[4](Natural Language Toolkit) disponible para el lenguaje de programación *Python*.

6.2.2. Adecuación de los métodos para clasificar

En muchas tareas de similitud textual se requiere llevar a cabo un proceso de clasificación. Si nos interesa saber si un par de textos son o no paráfrasis, son o no son plagio, etc, entonces el resultado del cálculo de similitud debe ser una etiqueta y no el valor de similitud.

Para poder llevar a cabo este tipo de clasificación, se necesita contar con al menos un umbral para saber si los resultados de las funciones de distancia de edición, alineación o

[4]http://www.nltk.org/

de la generada automáticamente con programación genética son o no relevantes[5] para la tarea en la que nos encontramos.

Para encontrar un buen umbral, en la etapa de entrenamiento, una vez que se haya encontrado el valor de similitud por medio de los métodos descritos en los capítulos 4 y 5, se propone recorrer los pares de texto de mayor valor a menor valor de similitud e ir moviendo el umbral de arriba abajo calculando la medida F-1 cada vez (ver ecuaciones 6.5, 6.6 y 6.7). Todos los valores que sean mayores que el umbral en turno se considerarán como relevantes mientras que los que estén por debajo se considerarán no relevantes. Al final el método se quedará con el umbral que haya generado la mejor medida F-1.

En la figura 6.1 se muestra el proceso que se lleva a cabo para clasificar. Primero se obtienen los valores de similitud del corpus de entrenamiento ya sea con los métodos de distancia de edición o con la generación automática de funciones (solo uno de ellos). Luego el proceso de encontrar el umbral que mejor particiona los resultados (en relevantes y no relevantes) empieza. Este proceso es iterativo y al finalizar el mejor umbral en el corpus de entrenamiento es encontrado. Luego se le aplica el mismo método que se utilizó en la fase de entrenamiento al corpus de prueba y con el umbral encontrado anteriormente se decide la etiqueta final de cada par de textos.

6.2.3. Métricas evaluación

Para la evaluación de un clasificador se tiene que cotejar la predicción del clasificador y la clase real de los objetos de evaluación. Para este análisis se utiliza una matriz de confusión; en la figura 6.2 se presenta el modelo general de una matriz de confusión. En las columnas se tiene la clase que el clasificador ha predicho y, en las filas, las clases a las que realmente pertenecen los objetos. En esta matriz de confusión se registra el número

[5]En este contexto, un texto es relevante si la etiqueta real en el corpus es positiva

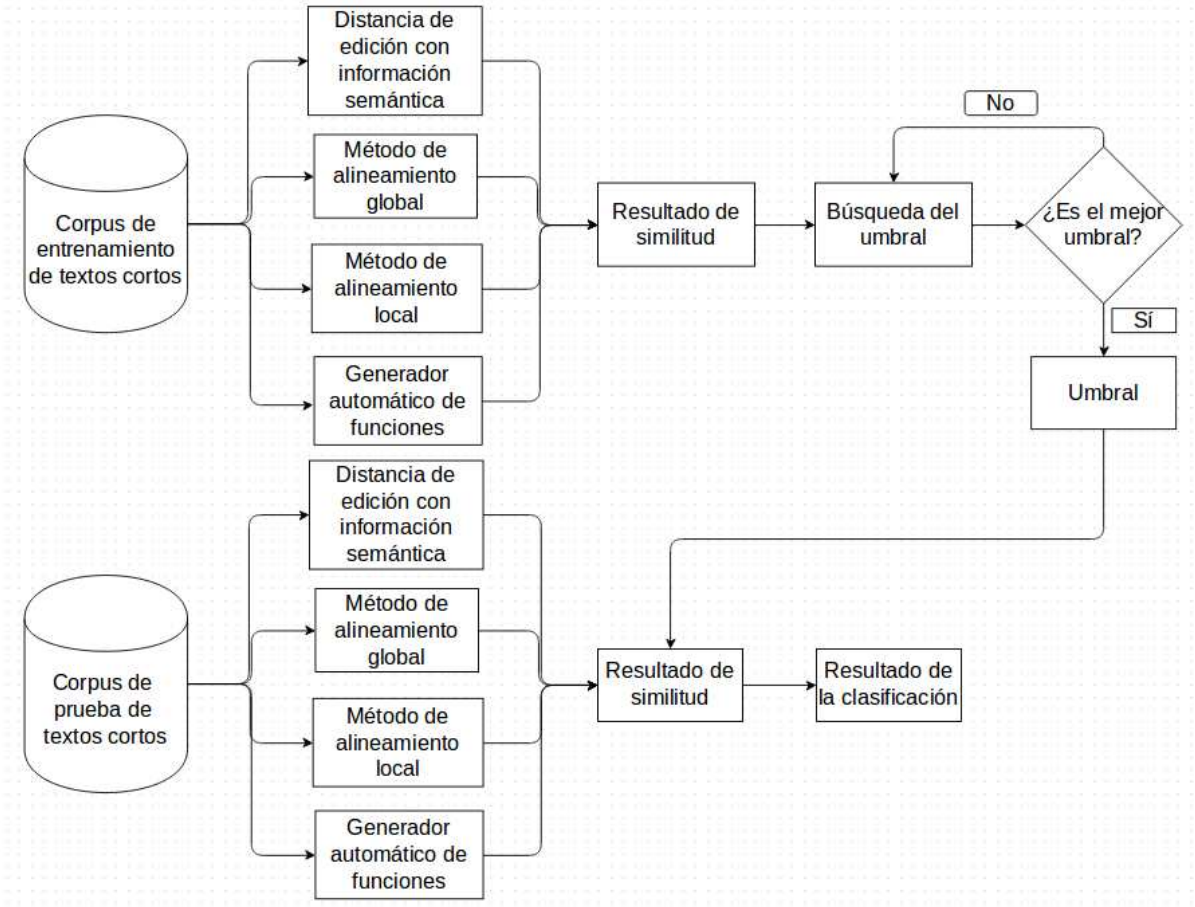

Figura 6.1: Diagrama para clasificar etiquetas discretas a partir de datos continuos

		Predicción	
		Positiva	Negativa
Real	Positiva	***TP***	***FN***
	Negativa	***FP***	***TN***

Figura 6.2: Modelo general de la matriz de confusión

de aciertos y errores (por cada clase) que tuvo el clasificador al ser evaluado con un conjunto de objetos particular. TP (true positives) son el número de aciertos de la clase positiva y TN (true negatives) los de la clase negativa. Por otro, lado FP (false positive) es el número de errores que tiene la clase positiva y FN (false negative) son de la clase negativa. Los errores por clase se refieren a los objetos que fueron predichos como cierta categoría por el clasificador, cuando en realidad son de otra (Kohavi & Provost, 1998).

Con estos datos, normalmente, se calculan dos medidas: precisión (ecuación 6.1) y recuerdo (ecuación 6.3). La precisión, P, es una medida que indica cuántos de los objetos fueron correctamente predichos en cierta clase. El recuerdo indica cuántos de los objetos de una cierta clase fueron reconocidos como tales.

$$P_{positiva} = \frac{TP}{TP + FP} \tag{6.1}$$

$$P_{negativa} = \frac{TN}{TN + FN} \tag{6.2}$$

$$R_{positiva} = \frac{TP}{TP + FN} \tag{6.3}$$

$$R_{negativa} = \frac{TN}{TN + FP} \tag{6.4}$$

Debido a que cada una de las medidas atiende a características distintas, ambas deseables, y para manejar el compromiso entre ellas, se ha utilizado la $F_1 - measure$, que involucra ambos aspectos. Al igual que para el recuerdo y la precisión existe una $F_1 - measure$ para cada clase (ecuación 6.5 y 6.6), y para tener un desempeño global en la evaluación de los métodos se ha utilizado el promedio de $F_1 - measure$ de las clases (ecuación 6.7)

$$F_1 - measure_{positiva} = \frac{2P_{positiva}R_{positiva}}{P_{positiva} + R_{positiva}} \tag{6.5}$$

$$F_1 - measure_{negativa} = \frac{2P_{negativa}R_{negativa}}{P_{negativa} + R_{negativa}} \tag{6.6}$$

$$F_1 - measure_{promedio} = \frac{F_1 - measure_{positiva} + F_1 - measure_{negativa}}{2} \tag{6.7}$$

6.2.4. Método de evaluación

Para evaluar todos los métodos en el corpus METER, se utilizará validación cruzada en k pliegues (k-folds cross validation) (Yu *et al.*, 2014), usualmente utilizada en la evaluación de los métodos de clasificación. Se emplearon 10 pliegues ($k = 10$).

Esta técnica consiste en una segmentación del corpus de evaluación en k partes no traslapadas y en realizar por separado las k clasificaciones. Este método permite que todos los ejemplos con los que se cuenta para la evaluación estén al menos una vez en el conjunto de entrenamiento y en el de la evaluación. El método permite aumentar la confianza de que los resultados no dependen del conjunto de evaluación particular ni del conjunto particular de entrenamiento.

Para poder utilizar los métodos, se aprenderá, el umbral que mejor divida los datos en

sus respectivas clases positivas y negativas. En el caso del corpus del MSRP se utilizará la parte de entrenamiento para llevar a cabo esta tarea, mientras que en el cospus del METER se evaluará con la técnica de validación cruzada (Yu *et al.*, 2014).

También se compararán los resultados obtenidos con algunos trabajos en el estado del arte que utilizan estas mismas colecciones.

6.3. Experimentos

6.3.1. Experimentos con las distancias de edición enriquecidas con información semántica

Para evaluar las medidas de distancias de edición propuestas en el capítulo 4 primero debemos comprobar si se mejora a los algoritmos originales. Esto para observar si la información semántica introducida está funcionando como se espera. Después estas mismas medidas se compararán con las medidas con mejores resultados en el estado del arte de cada corpus. En esta misma sección se presentarán los resultados obtenidos en cada experimento.

Objetivo

El objetivo de este experimento es, en primer lugar, evidenciar el mejoramiento al introducir información semántica a las medidas de distancias de edición. Posteriormente, se observará el comportamiento de cada una de estas medidas en las tareas de detección de plagio y paráfrasis para observar cuál de estas medidas funciona mejor dependiendo de la tarea y qué tan bien compiten con los mejores métodos en el estado del arte.

Baselines

Para esta sección, primero se obtendrán los resultados de los métodos originales (Distancia de edición, algoritmo de Neddleman-Wunsch y algoritmo de Smith-Waterman). En el caso de los algoritmos de alineación de secuencias las matrices de información serán binarias, es decir, si el par de palabras que se están comparando son iguales el valor en la matriz será de uno, en caso contrario, será de uno negativo. Esto para que puedan ser utilizados como *baselines*. En caso de mejorar los resultados que se obtengan de estos métodos con las extensiones propuestas en este trabajo, se daría evidencia de que la información semántica con la que se enriquecen estos métodos está funcionando como se espera.

Resultados

En las tablas 6.1 y 6.2 se presentan los resultados de las medidas originales (distancia de Levenshtein, algoritmo Neddleman-Wunsch, algoritmo Smith-Waterman). En las tablas se muestran cada uno de los métodos originales y su resultado de exactitud y el resultado de la medidad F-1 respectivamente obtenido en cada una de las dos colecciones de datos con los que se está experimentando, uno en cada columna de cada tabla.

En la tabla 6.3 se muestran los resultados de exactitud de cada uno de los métodos propuestos en el capítulo 4 con respecto a las colecciones MSRP y METER respectivamente.

Mientras que en la tabla 6.4 se muestran los resultados de la medidad F-1 para cada corpus con las medidas modificadas con información semantica.

En las figuras 6.3 y 6.4 se comparan seis resultados obtenidos a partir de las colecciones MSRP y METER, de izquierda a derecha, el primer par de resultados representa los valores obtenidos por la distancia de Levenshtein y sus modificación, el segundo par

Tabla 6.1: Resultados de exactitud de los métodos originales sin información semántica

Método	**MSRP**	**METER**
Distancia de Levenshtein	0.30	0.25
Algoritmo Neddleman-Wunsch	**0.34**	0.28
Algoritmo Smith-Waterman	0.28	**0.31**

Tabla 6.2: Resultados de la medida F-1 de los métodos originales sin información semántica

Método	**MSRP**	**METER**
Distancia de Levenshtein	0.28	0.27
Algoritmo Neddleman-Wunsch	**0.33**	0.26
Algoritmo Smith-Waterman	0.29	**0.32**

Tabla 6.3: Resultados de exactitud de los métodos modificados con información semántica

Método enriquecido con información semántica	**MSRP**	**METER**
Distancia de Levenshtein semántica	0.68	0.58
Algoritmo Neddleman-Wunsch semántico	**0.73**	0.62
Algoritmo Smith-Waterman semántico	0.70	**0.65**

Tabla 6.4: Resultados de la medida F-1 de los métodos modificados con información semántica

Método enriquecido con información semántica	**MSRP**	**METER**
Distancia de Levenshtein semántica	0.72	0.60
Algoritmo Neddleman-Wunsch semántico	**0.78**	0.62
Algoritmo Smith-Waterman semántico	0.74	**0.66**

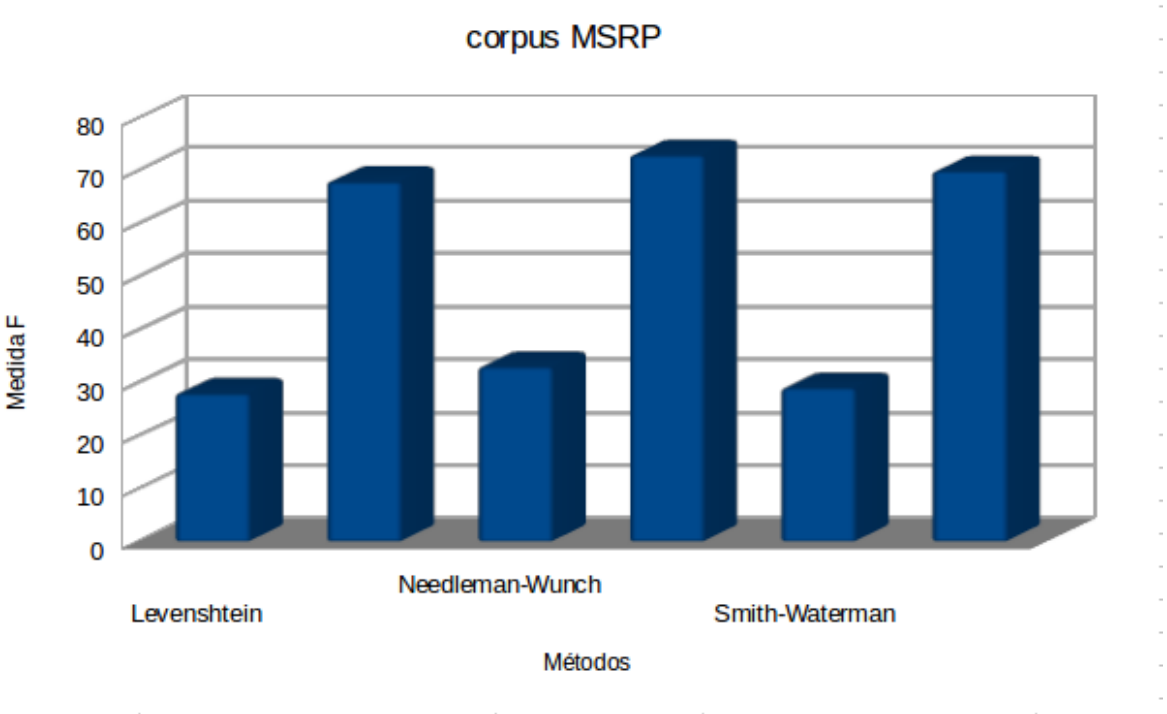

Figura 6.3: Comparación de las distancias de edición originales con sus modificaciones en el corpus MSRP

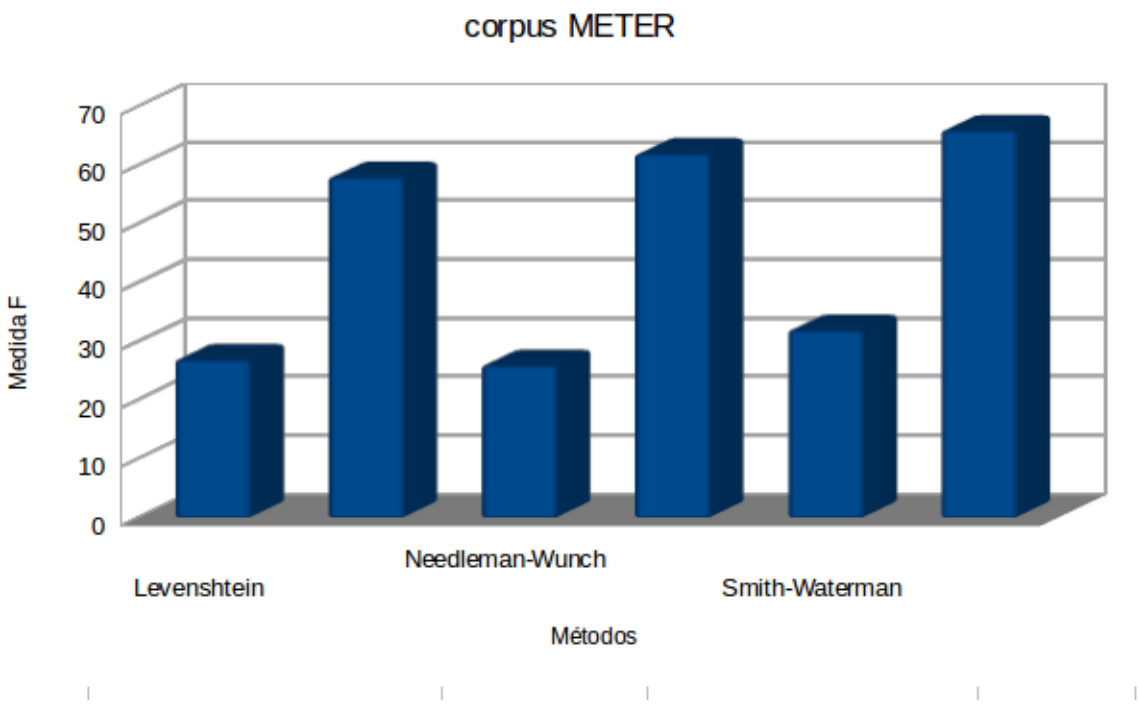

Figura 6.4: Comparación de las distancias de edición originales con sus modificaciones en el corpus METER

representa al algoritmo Needleman-Wunsch y su modificación, finalmente, el último par de resultados representan los valores obtenidos por el algoritmo Smith-Waterman y su respectiva modificación.

Discusión de los resultados

Los resultados de las tablas 6.1 y 6.2 representan los valores de exactitud y medida F-1 obtenidos a partir de las colecciones MSRP y METER. Estos resultados pueden ser consisderados bajos ya que ninguno de ello supera el 0.35 de exactitud o medida F-1 en ninguna de las colecciones.

En el corpus de detección de paráfrasis el mejor resultado lo obtiene el algoritmo de Needleman-Wunsch (alineamiento global), tanto en exactitud como en medida F-1 donde obtuvo 0.34 y 0.33 respectivamente. En este mismo corpus el algoritmo de Smith-Waterman obtuvo resultados más bajos que los de Needleman-Wunsch aunque mejor que los de Levenshtein donde en medida F-1 lograron 0.30 y 0.28.

En el corpus de detección de plagio el método con mejor medida F-1, a diferencia del corpus anterior, fue el algoritmo de Smith-Waterman (alineamiento local) donde obtuvo 0.31 de exactitud y 0.32 de medida F-1. De nueva cuenta la distancia de edición obtuvo los resultados más bajos con 0.25 y 0.27 de exactitud y medida F-1 respectivamente. En esta tarea el algoritmo de Needleman-Wunsch quedó por debajo del algoritmo de Smith-Waterman obteniendo 0.28 y 0.26 de exactitud y medida F-1.

Cuando se experimentó con las distancias con información semántica se nota una mejoría considerable en ambas tareas. Al comparar la medida F-1 de los algoritmos originales con sus modificaciones se puede ver que casi se duplican los resultados.

El algoritmo de Levenshtein en detección de paráfrasis y plagio obtiene 0.28 y 0.27 respectivamente mientras que su modificación obtiene 0.72 y 0.60, es decir, en la tarea

de detección de paráfrasis mejoró 0.44 la medida F-1 mientras que en detección de plagio la mejoró en 0.33, es decir, 157 % y 122 % de avance en cada tarea.

La modificación de Needleman-Wunsch obtuvo de medida F-1 0.33 y 0.26 en las tareas de paráfrasis y plagio mientras que su modificación obtuvo 0.78 y 0.62 mejorando en 0.45 y 0.46, lo que equivale al 136 % y 176 % de mejora respectivamente.

Por su parte, el algoritmo de Smith-Waterman obtuvo 0.29 y 0.32 en paráfrasis y plagio y su modificación 0.74 y 0.66 mejorando en 0.45 y 0.34 de medida F-1; lo que representa el 155 % y 106 % de progreso respectivamente.

Los porcentajes indican una notable mejoría en los resultados. Todas las mejoras están por encima del 100 %, es decir, todos los resultados son mejorados por más del doble. La mejora más significativa surgió al aplicar la modificación del algoritmo de Needleman-Wunsch sobre el cropus de detección de plagio.

De las tres modificaciones, la que mejor funcionó en la tarea de detección de paráfrasis fue la del algoritmo de Needleman-Wunsch obteniendo 0.78 mientras que el algoritmo de Smith-Waterman obtuvo 0.74. El caso contrario ocurre cuando se trata de la tarea de detección de plagio donde el mejor resultado de la medida F-1 la obtiene Smith-Waterman con 0.66 mientras que Needleman-Wunsch obtuvo 0.62. Esto da evidencia de que un alineamiento de naturaleza global se apega mejor a tareas como la detección de paráfrasis por como se lleva a cabo este fenómeno, por otro lado, un algoritmo de alineamiento local funciona mejor para tareas como la detección de plagio probando de cierto modo que buscando en localidades del texto es más probable poder atacar mejor esta tarea.

Costo de las modificaciones

Los resultados presentados hasta ahora sugieren que las modificaciones están funcionando bien. Algo importante que se debe discutir es el costo de estas mejoras.

Si bien las complejidades de los tres algoritmos no aumenta, es decir, siguen en el orden de $O(n^2)$ (Cormen *et al.*, 2001), el tiempo del procesador sí se ve afectado por estas modificaciones.

Para entender un poco cuál es el costo de introducir la información semántica de *WordNet* se calculó el tiempo de comparar 640460 pares de palabras; éstas fueron extraídas del corpus de entrenamiento del MSRP. El experimento se corrió en una computadora con un sistema operativo Ubuntu en su versión 12.04 de 64 *bits* con memoria RAM de 32 GB y un procesador intel Core i7 a 3.4GHz.

En este equipo, para terminar de procesar los 640460 pares de palabras se consumió un total de 279.9204 segundos, es decir, alrededor de 4.6 minutos. En promedio, por cada par de palabras el equipo demoró 0.4 *ms*.

En conclusión, los algoritmos modificados tardarán un promedio de 0.4 *ms* por cada par de palabras que se analicen semánticamente en *WordNet* con un equipo de características similares a las reportadas en esta sección, lo cual no parece ser un costo muy alto tomando en cuenta las mejoras se alcanzaron.

6.3.2. Experimentos con la generación automática de funciones

Objetivo

Lo primero que se debe comprobar para este método es que el todo sea mejor que cada una de sus partes, es decir, que cada una de las medidas que se utilizó para llegar a una función general no sea mejor que el resultado final. También se debe evidenciar que

esta mezcla es mejor que alguna otra mezcla más sencilla utilizando las mismas medidas. Esto para probar que vale la pena hacer esta mezcla en lugar de solo utilizar algunas de las partes.

Ya que los resultados de un algoritmo de programación genética no son deterministas se correrá el algoritmo de generación automática de funciones cinco veces para comparar los diferentes resultados. En las cinco corridas se utilizarán los mismos parámetros que están definidos en la tabla 5.1.

Baselines

Para los experimentos de esta sección, como *baselines* se propone comparar los métodos de forma individual y algunas combinaciones más sencillas que la de la programación genética propuesta. Estas combinaciones serán un promedio aritmético, una regresión lineal simple y con máquina de vectores de soporte[6].

Resultados

En la tabla 6.5 se muestran los resultados de los métodos que sirven para medir el grado de traslape de conjuntos de palabras.

Tabla 6.5: Resultados de la medida F-1 de los métodos que consideran traslape de conjuntos de palabras

Método	**MSRP**	**METER**
Coeficiente de Dice	**0.50**	**0.41**
Coeficiente de Jaccard	0.32	0.39
Coeficiente de traslape	0.45	0.37
Coeficiente de coseno	0.28	0.25

En la tabla 6.6 se muestran los resultados obtenidos de la medida F-1 de las medidas individuales de la sección de medidas basadas en secuencias de palabras.

[6]Con los parámetros que tiene por *default* Weka

Tabla 6.6: Resultados de la medida F-1 de los métodos que consideran la secuencia de palabras

Método	**MSRP**	**METER**
Subsecuencia común más larga	0.45	0.38
Subsecuencia común más larga en palabras vacías	0.38	**0.44**
Algoritmo de Levenshtein	0.28	0.33
Algoritmo de Levenshtein en palabras vacías	0.31	0.30
Coeficiente de Dice con bi-gramas	**0.53**	0.40
Coeficiente de Jaccard con bi-gramas	0.37	0.39
Coeficiente de traslape con bi-gramas	0.48	0.40
Similitud coseno con bi-gramas	0.35	0.38
Algoritmo de posicionamiento de palabras	0.42	0.38

En la tabla 6.7 se muestran los resultados obtenidos de las medidas de detección de similitud a partir de información semántica.

Tabla 6.7: Resultados de la medida F-1 de los métodos que consideran información semántica

Método	**MSRP**	**METER**
Coeficiente de Dice con información semántica	0.65	0.42
Coeficiente de Dice con información semántica clusterizada	**0.69**	**0.47**
Coeficiente de Dice por generalidad basada en conocimiento	0.64	0.41
Coeficiente de Dice por generalidad basada en corpus	0.65	0.43

En la tabla 6.8 se presentan los resultados de tres combinaciónes de los métodos propuestos basados en traslape de conjuntos de palabras, secuencia de palabras e información semántica. Estas combinaciones son: un promedio aritmético de todas las medidas, una regresión lineal y utilizando una máquina de vectores de soporte.

En la tabla 6.9 se presentan los resultados de la medida F-1 de cinco corridas del algoritmo de programación genética sobre los dos corpus con lo que se trabaja.

Tabla 6.8: Resultados de la medida F-1 de los métodos combinados

Método	**MSRP**	**METER**
Promedio artimético	0.60	0.37
Regresión lineal	0.71	0.46
Máquina de vectores de soporte	**0.80**	**0.67**

Tabla 6.9: Resultados de la medida F-1 de la combinación mediante programación genética junto con sus resultados de la función de aptitud

Método	**MSRP**	**Fitness en MSRP**	**METER**	**Fitness en METER**
Experimento 1	0.83	0.81	0.69	0.79
Experimento 2	**0.86**	0.84	**0.73**	0.80
Experimento 3	0.84	0.83	0.71	0.77
Experimento 4	0.82	0.83	0.70	0.77
Experimento 5	0.83	0.84	0.69	0.79
Promedio	0.836	0.83	0.704	0.784
Desviación estándar	0.01517	0.01225	0.01673	0.01342

Discusión de los resultados

En los resultados presentados en esta sección se puede ver en la tabla 6.5 que de los cuatro coeficientes que se utiliza para medir el traslape de la palabras, el que mejor resultados obtiene en las tareas de detección de paráfrasis y plagio es el del coeficiente de Dice obteniendo 0.50 y 0.41 de medida F-1 en estas tareas respectivamente.

En la parte de las medidas basadas en secuencia de palabras no fue el mismo método el que obtuvo el mejor resultado en ambas tareas a diferencia del coeficiente de Dice. Para estos métodos, en la tarea de detección de paráfrasis el coeficiente de Dice con bigramas obteniendo 0.53 de medida F-1 superando al coeficiente de Dice original en esta tarea. Por otro lado, en la tarea de detección de plagio, el método con mejor resultado de medida F-1 fue el de la subsecuencia común más larga en palabras vaciás con 0.44 y de igual forma supera lo que obtuvo el coefriciente de Dice en el corpus de METER.

Para la categoría de las medidas que toman en cuenta información semántica se puede

ver en la tabla 6.7 que la mejor medida tanto en la tarea de detección de parárafris como de plagio es la del coeficiente de Dice con información semántica clusterizada con 0.69 y 0.47 respectivamente superando a las mejor medida de secuencia de palabras y traslape de conjuntos de palabras lo que podría dar indicios de que la información semántica es de mayor ayuda en estas colecciones de datos.

Para las combinaciones alternativas de la tabla 6.8 se puede ver que la mejor combinación es la que se lleva a cabo a partir de una máquina de vectores de soporte obteniendo 0.80 y 0.67 en cada una de las tareas y superando al promedio aritmetico y regresión lineal además de superar a todas las medidas de manera individual.

Como se observa en la tabla 6.9 el experimento del generador automático de funciones más bajo en ambas tareas obtiene mejores resultados que los mejores resultados de cada medida de forma individual y de sus combinaciones. El peor resultado de los cinco experimentos obtiene 0.82 de medida F-1 en la tarea de detección de paráfrasis mientras que la mejor medida individual había sido la de la medida de Needleman-Wunsch de la tabla 6.4 donde obtiene 0.78, además de superar a la mejor combinación que fue la de la máquina de vectores de soporte la que obtuvo 0.80. Lo mismo ocurre en la tarea de detección de plagio, donde el peor resultado de los cinco experimentos obtiene 0.73 cuando la mejor medida indivudual fue la medida de Smith-Waterman con 0.66 de medida F-1; y de igual forma la máquina de vectores de soporte obtuvo de nueva cuenta la mejor medida F-1 de las combinaciones simples con 0.67. Por otro lado, al hacer los cinco experimentos del algoritmo de programación genética se puede ver que los valores de desviación estándar obtenidos de las dos colecciones son pequeños, los que nos da una idea de que este método es congruente con los resultados, es decir, no son resultados muy diferentes entre sí.

En la imagen 6.5 Se muestra la comparación del mejor y peor resultado obtenido a

través de programación genética con los mejores resultados obtenido por categoría en el corus MSRP. Para este corpus la mejores medidas fueron: en traslape de conjuntos de palabras el coeficiente de Dice, para secuencia de palabras fue el coeficiente de Dice con bigramas, para la categoría de métodos con información semántica el mejor resultado fue el de la medida de Needleman-Wunsch con información semántica. Además de compararse con el resultado de la máquina de soporte vectorial.

En la imagen 6.6 Se muestra la comparación del mejor y peor resultado obtenido a través de programación genética con los mejores resultados obtenido por categoría en el corpus METER. Para este corpus la mejores medidas fueron: en traslape de conjuntos de palabras el coeficiente de Dice, para secuencia de palabras fue la subsecuencia común más larga en palabras vacías, para la categoría de métodos con información semántica el mejor resultado fue el de la medida de Smith-Waterman con información semántica. Además de compararse con el resultado de la máquina de soporte vectorial.

6.4. Comparación general

Tabla 6.10: Resultados generales del corpus MRSP

Método	**Exactitud**	**F-1**
Mihalcea *et al.* (2006)	0.65	0.75
Fernando & Stevenson (2008)	0.74	0.81
Madnani *et al.* (2012)	0.77	0.84
Distancia de Levenshtein semántica	0.68	0.72
Algoritmo Neddleman-Wunsch semántico	0.73	0.78
Algoritmo Smith-Waterman semántico	0.70	0.74
Mejor experimento con programación genética	**0.82**	**0.86**
Peor experimento con programación genética	0.77	0.82

En la tabla 6.10 se muestran los resultados de los mejores trabajos en el estado del arte que utilizan el corpus MSRP. También se muestran los resultados de las distancias

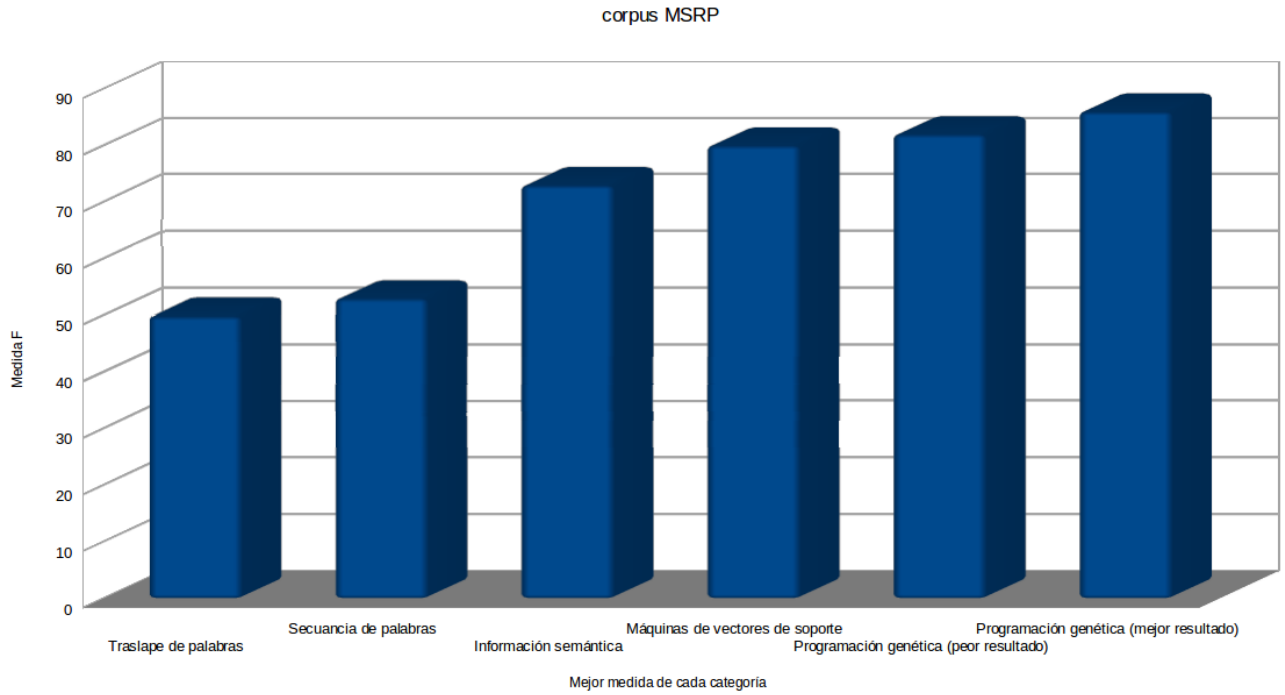

Figura 6.5: Comparación de las mejores medidas por categorías del corpus MSRP

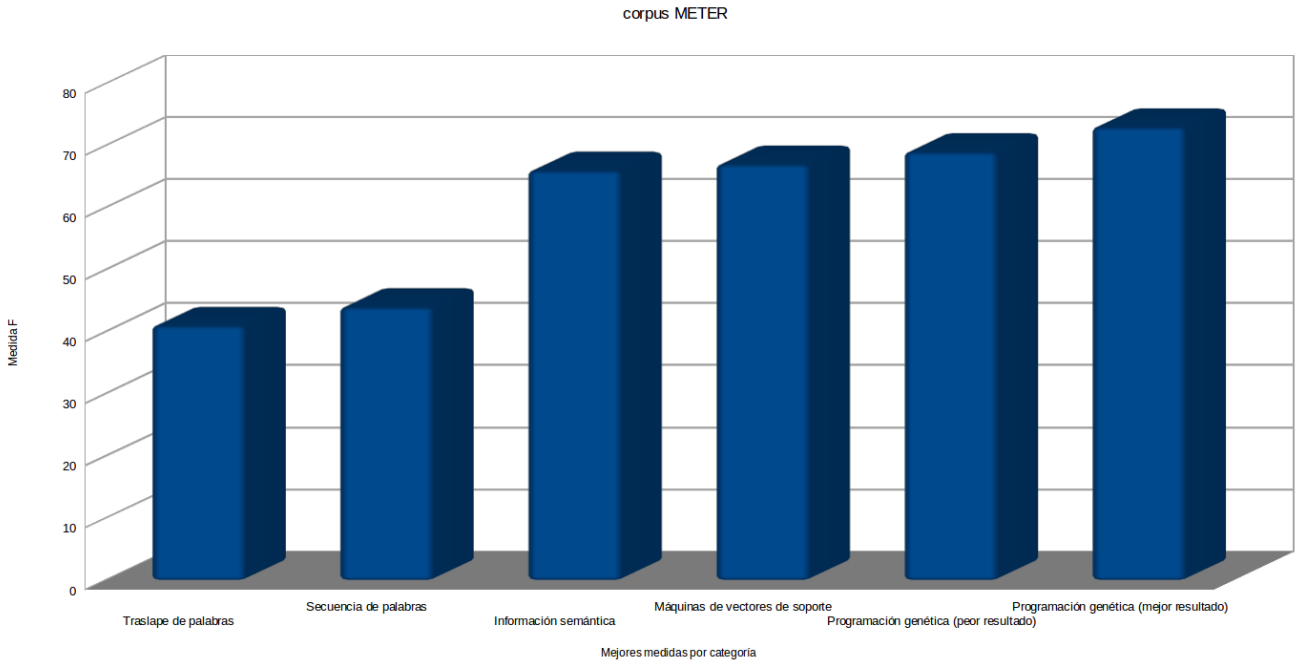

Figura 6.6: Comparación de las mejores medidas por categorías del corpus METER

Tabla 6.11: Resultados generales del corpus METER

Método	**Exactitud**	**F-1**
Chong *et al.* (2010)	0.70	0.69
Sánchez-Vega *et al.* (2013)	0.78	0.69
Distancia de Levenshtein semántica	0.58	0.60
Algoritmo Neddleman-Wunsch semántico	0.62	0.62
Algoritmo Smith-Waterman semántico	0.65	0.66
Mejor experimento con programación genética	**0.79**	**0.73**
Peor experimento con programación genética	0.71	0.69

de edición enriquecidas con información semántica y los resultados de la peor y mejor corrida del algoritmo de programación genética. En esta tabla se puede ver que tanto en exactitud como en medida F-1, los mejores resultados los obtiene el algoritmo de programación genética.

De igual forma, en la tabla 6.11 los resultados se comparan con los mejores trabajos hasta el momento en el estado del arte. En este caso, el mejor resultado lo obtiene la mejor corrida de programación genética, sin embargo, el peor resultado obtiene resultados muy similares a los de los trabajos de Chong *et al.* (2010) y Sánchez-Vega *et al.* (2013).

De las distancias de edición enriquecidas semánticamente se puede ver que no obtienen los mejores resultados de la tabla, sin embargo, en el corpus de detección de paráfrasis la modificación del algoritmo de Needleman-Wunsch obtiene mejores resultados que el trabajo de Mihalcea *et al.* (2006) aunque se ve superado por los trabajos de Fernando & Stevenson (2008) y Madnani *et al.* (2012). Por otro lado, en la tarea de detección de plagio tampoco logra superar a los trabajos de Chong *et al.* (2010) y Sánchez-Vega *et al.* (2013) pero se debe tomar en cuenta que estas distancias trabajan de forma no supervisada (aunque para este trabajo se tuvo que llevar a cabo una fase de entrenamiento, esto porque se necesitaba clasificar de forma binaria y no continua; así que era necesario encontrar un umbral) lo cual podría ser útil cuando no se pueda llevar a cabo la fase de

entrenamiento para clasificar.

Capítulo 7

Conclusiones y trabajo a futuro

7.1. Recapitulación

Para llevar a cabo la investigación presentada en este trabajo, se profundizó en el tema de detección de similitud textual realizando una descripción de la tarea y sus características tanto de desarrollo como de evaluación. Se llevó a cabo la revisión del estado del arte de este tema y se mostró una clasificación para mostrar los trabajos existentes.

Se presentaron dos enfoques en este trabajo, el primer enfoque fue el de las distancias de edición informadas semánticamente, estas distancias fueron la distancia de Levenshtein y los algoritmos de alineación global y local (Needleman-Wunsch y Smith-Waterman), mientras que, el segundo enfoque fue el del método de generación automática de funciones, en este método se propusieron un conjunto de métodos individuales y un método que los conjunta a través de programación genética.

Se presentó un capítulo experimental donde se presentan los resultados de cada uno de estos métodos. Estas propuestas se evaluaron en las tareas de detección de paráfrasis y plagio ya que por las características de estos métodos se prestaban para ser evaluados

en estas tareas.

Finalmente, en este capítulo se presentan las conclusiones de los métodos y sus resultados en la parte de experimentación. Además de propuestas futuras para continuar con este trabajo.

7.2. Conclusiones

Después de llevar a cabo este trabajo, finalmente se puede concluir que:

- Los métodos basados en algoritmos de alineación de secuencias biológicas al introducirles información semántica, se puede observar una notable mejoría con respecto a los originales. Esto sugiere que la información añadida esta funcionando correctamente y que las colecciones de datos utilizadas contienen diversas transformaciones semánticas. Esta mejoría se debe a que estos métodos permiten capturar cambios tanto léxicos como semánticos, mientras que los métodos originales no lo permiten. Además que la estrategia global permite detectar similitud a lo largo de ambos textos comparados mientras que la estrategia local se comporta mejor cuando solo pequeñas fracciones de los textos son similares.

- De las distancias de edición modificadas, el método de Needleman-Wunsch con información semántica obtiene mejores resultados con la tarea de detección de paráfrasis, mientras que el método Smith-Waterman obtiene mejores resultados en la tarea de detección de plagio. Esto indica que las características de alineación global y local respectivamente funciona mejor en dependencia de la tarea con que se trabaja. Alinear de forma global funciona mejor en paráfrasis porque al generar paráfrasis se incluye prácticamente todo el texto pero con sus transformaciones semánticas, por lo que alinear de forma global para encontrar similitud funciona

mejor. Por otro lado al generar plagio solo se incluyen peuqeñas porciones del texto original para que el texto plagiado sea lo menos sospechoso posible, por lo que alinear de forma local funciona mejor.

- En el método de programación genética para generar funciones de manera automática, se puede observar que esta forma de combinar las distintas métricas resultó adecuada, pues superó los resultados de las métricas individuales y de otras combinaciones. Además es estable porque los resultados no varian mucho al repetir los experimentos, por esta razón se podría hablar de que este método puede llegar a un buen resultado en pocas repeticiones.

- El generador automático de funciones obtiene mejores resultados que todas las medidas incluidas de forma individual y que algunas combinaciones más simples como un promedio aritmético, regresión lineal y una máquina de vectores de soporte. Lo que da evidencia que la propuesta de combinar medidas a través de programación genética está funcionando de buena forma.

- Al observar los resultados podemos concluir que el método del generador automático de funciones mantiene su competitividad en las tareas de detección de plagio y detección de paráfrasis con respecto a los trabajos en el estado del arte. donde incluso se supera a los mejores trabajos de cada tarea. Esto podría indicar que este método mantiene un buen funcionamiento sin importar la tarea en la que se aplique. Esto se debe a que en cada tarea este método obtiene una función especializada y por ello resultados que superan a los métodos del estado del arte.

7.3. Trabajo a futuro

Después del trabajo realizado se abren algunos posibles trabajos a desarrollar como:

- Aplicar los métodos propuestos a más tareas donde lo primordial sea la detección de similitud textal, estas tareas podrían ser: evaluación de resúmenes automáticos, evaluación de traducciones automáticas, clasificaciones de tiuters, clasificación temática, etc.

- Introducir información semántica a los métodos propuestos desde otras ontologías como Babelnet o Eurowordnet para ver si los resultados pueden mejorar con alguna de estas ontologías.

- Llevar a cabo un proceso de clasificación cruzada, es decir, entrenar el algoritmo de programación genética con datos de alguna tarea y probar la función resultante con datos de prueba de otra tarea. Esto para ver qué tan bien funcionaría una funcíon especializada de una tarea en otra.

- Profundizar en los operadores de programación genética para modificar algunos existentes o proponer nuevos que se adapten mejor a la tarea de detección de similitud a través de programación genética.

Referencias

Khaled Abdalgader. Word sense identification improves the measurement of short-text similarity. In *The International Conference on Computing Technology and Information Management (ICCTIM2014)*, páginas 233–243. The Society of Digital Information and Wireless Communication, 2014.

Palakorn Achananuparp, Xiaohua Hu, & Xiajiong Shen. The evaluation of sentence similarity measures. In *Data Warehousing and Knowledge Discovery*, páginas 305–316. Springer, 2008.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, & Weiwei Guo. sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity. In *SEM 2013: The Second Joint Conference on Lexical and Computational Semantics. Association for Computational Linguistics*. Citeseer, 2013.

Mohammad Yahya H Al-Shamri. Power coefficient as a similarity measure for memory-based collaborative recommender systems. *Expert Systems with Applications*, páginas 5680–5688, 2014.

Miguel Álvarez, Ariel Carrasco, & Francisco Martínez. Combining techniques to find the number of bins for discretization. In *Proceedings of the V Chilean Workshop on Pattern Recognition*, páginas 520–526. IEEE, 2013.

Robin Aly, Dolf Trieschnigg, Kevin McGuinness, Noel E OConnor, & Franciska de Jong. Average precision: Good guide or false friend to multimedia search effectiveness? In *MultiMedia Modeling*, páginas 239–250. Springer, 2014.

Salha Alzahrani & Naomie Salim. Fuzzy semantic-based string similarity for extrinsic plagiarism detection. *Braschler and Harman*, 2010.

Shahab Araghinejad. Support vector machines. In *Data-Driven Modeling: Using MATLAB® in Water Resources and Environmental Engineering*, páginas 195–211. Springer, 2014.

Meghan J Babcock, Vivian P Ta, & William Ickes. Latent semantic similarity and language style matching in initial dyadic interactions. *Journal of Language and Social Psychology*, 33(1):78–88, 2014.

Thomas Back, David B Fogel, & Zbigniew Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

Benjamin Balsmeier, Gabe Fierro, Lee Fleming, Kevin Johnson, Aditya Kaulagi, Guan-Cheng Li, & William Yeh. Weekly disambiguations of us patent grants and applications. 2014.

Carmen Banea, Yoonjung Choi, Lingjia Deng, Samer Hassan, Michael Mohler, Bishan Yang, Claire Cardie, Rada Mihalcea, & Janyce Wiebe. Cpn-core: A text semantic similarity system infused with opinion knowledge. *Atlanta, Georgia, USA*, página 221, 2013.

Daniel Bär, Chris Biemann, Iryna Gurevych, & Torsten Zesch. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the Sixth International Workshop on Semantic Evaluation*, páginas 435–440. Association for Computational Linguistics, 2012.

Alberto Barrón-Cedeno, Paolo Rosso, Eneko Agirre, & Gorka Labaka. Plagiarism detection across distant language pairs. In *Proceedings of the 23rd International Conference on Computational Linguistics*, páginas 37–45. Association for Computational Linguistics, 2010.

Rahul Bhagat & Eduard Hovy. What is a paraphrase? 2013.

Arpit Bhardwaj, Aruna Tiwari, M Vishaal Varma, & M Ramesh Krishna. Classification of eeg signals using a novel genetic programming approach. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, páginas 1297–1304. ACM, 2014.

Antoine Bordes, Xavier Glorot, Jason Weston, & Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

Johan Bos. Recognizing textual entailment and computational semantics. In *Computing meaning*, páginas 89–105. Springer, 2014.

Alexander Budanitsky & Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

Dieu Tien Bui, Tien Chung Ho, Inge Revhaug, Biswajeet Pradhan, & Duy Ba Nguyen. Landslide susceptibility mapping along the national road 32 of vietnam using gis-based j48 decision tree classifier and its ensembles. In *Cartography from Pole to Pole*, páginas 303–317. Springer, 2014.

C Cajías. Quines corollary on analysis and his notion of paraphrase. *Episteme NS*, 33 (1), 2014.

Julio Castillo & Paula Estrella. Semantic textual similarity for mt evaluation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, páginas 52–58. Association for Computational Linguistics, 2012.

Dariusz Ceglarek. Evaluation of the shapd2 algorithm efficiency in plagiarism detection tasks. In *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on*, páginas 465–470. IEEE, 2013.

Yee Seng Chan & Hwee Tou Ng. Maxsim: A maximum similarity metric for machine translation evaluation. In *ACL*, páginas 55–62, 2008.

Wui Lee Chang, Kai Meng Tay, & Chee Peng Lim. A new evolving tree for text document clustering and visualization. In *Soft Computing in Industrial Applications*, páginas 141–151. Springer, 2014.

Surajit Chaudhuri, Venkatesh Ganti, & Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, páginas 5–5. IEEE, 2006.

Huang Cheng-Hui, Yin Jian, & Hou Fang. A text similarity measurement combining word semantic information with tf-idf method. *Chinese journal of computers*, 34(5): 856–864, 2011.

Peter A Chew. Automated account reconciliation method, January 28 2014. US Patent 8,639,596.

Miranda Chong, Lucia Specia, & Ruslan Mitkov. Using natural language processing for automatic detection of plagiarism. In *Proceedings of the 4th International Plagiarism Conference (IPC 2010), Newcastle, UK*, 2010.

Gobinda Chowdhury. *Introduction to modern information retrieval*. Facet publishing, 2010.

Souvik Dutta Chowdhury, Ujjwal Bhattacharya, & Swapan K Parui. Levenshtein distance metric based holistic handwritten word recognition. In *Proceedings of the 4th International Workshop on Multilingual OCR*, página 17. ACM, 2013.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, *et al. Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

Lieven Decock & Igor Douven. Similarity after goodman. *Review of philosophy and psychology*, 2(1):61–75, 2011.

Michael Denkowski & Alon Lavie. Extending the meteor machine translation evaluation metric to the phrase level. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, páginas 250–253. Association for Computational Linguistics, 2010.

George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, páginas 138–145. Morgan Kaufmann Publishers Inc., 2002.

Bill Dolan, Chris Quirk, & Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*, página 350. Association for Computational Linguistics, 2004.

James Dougherty, Ron Kohavi, Mehran Sahami, *et al.* Supervised and unsupervised discretization of continuous features. In *ICML*, páginas 194–202, 1995.

Ofer Egozi, Shaul Markovitch, & Evgeniy Gabrilovich. Concept-based information retrieval using explicit semantic analysis. *ACM Transactions on Information Systems (TOIS)*, 29(2):8, 2011.

Anthony Fader, Luke S Zettlemoyer, & Oren Etzioni. Paraphrase-driven learning for open question answering. In *ACL (1)*, páginas 1608–1618, 2013.

Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

Xinyuan Feng, Jianguo Wei, Wenhuan Lu, & Jianwu Dang. Word semantic similarity calculation based on domain knowledge and hownet. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 12(2):1143–1148, 2014.

Samuel Fernando & Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*, páginas 45–52, 2008.

Evgeniy Gabrilovich & Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, volume 7, páginas 1606–1611, 2007.

Robert Gaizauskas, Jonathan Foster, Yorick Wilks, John Arundel, Paul Clough, & Scott Piao. The meter corpus: A corpus for analysing journalistic text reuse. páginas 214–223, 2001.

David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning*, volume 412. Addison-wesley Reading Menlo Park, 1989.

Wael H Gomaa & Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.

Nizar Habash & Ahmed Elkholy. Sepia: surface span extension to syntactic dependency precision-based mt evaluation. In *Proceedings of the NIST metrics for machine translation workshop at the association for machine translation in the Americas conference, AMTA-2008. Waikiki, HI*, 2008.

Bo Han & Timothy Baldwin. Lexical normalisation of short text messages: Makn sens a# twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, páginas 368–378. Association for Computational Linguistics, 2011.

Philip W Hedrick. Genetic similarity and distance: comments and comparisons. *Evolution*, páginas 362–366, 1975.

Graeme Hirst & David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database*, 305:305–332, 1998.

Chukfong Ho, Masrah Azrifah Azmi Murad, Rabiah Abdul Kadir, & Shyamala C Doraisamy. Word sense disambiguation-based sentence similarity. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, páginas 418–426. Association for Computational Linguistics, 2010.

V Melissa Holland, Michelle R Sams, & Jonathan D Kaplan. *Intelligent language tutors: Theory shaping technology*. Routledge, 2013.

Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, páginas 49–56, 2008.

Md Shafiqul Islam, MA Hannan, Hassan Basri, Aini Hussain, & Maher Arebey. Solid waste bin detection and classification using dynamic time warping and mlp classifier. *Waste Management*, 34(2):281–290, 2014.

S Jayaprada, P Prasad, S Vasavi, & I Ramesh Babu. Vizsfp: A visualizer for semantically similar frequent patterns in dynamic datasets. In *Advance Computing Conference (IACC), 2014 IEEE International*, páginas 422–427. IEEE, 2014.

Jay J Jiang & David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

Sergio Jimenez, Fabio Gonzalez, & Alexander Gelbukh. Text comparison using soft cardinality. In *String Processing and Information Retrieval*, páginas 297–302. Springer, 2010.

Allison B Kaufman, Erin N Colbert-White, & Curt Burgess. Higher-order semantic structures in an african grey parrots vocalizations: evidence from the hyperspace analog to language (hal) model. *Animal cognition*, 16(5):789–801, 2013.

Ishwinder Kaur & Anthony J Hornof. A comparison of lsa, wordnet and pmi-ir for predicting user click behavior. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, páginas 51–60. ACM, 2005.

R Kohavi & F Provost. Confusion matrix. *Machine learning*, 30(2-3):271–274, 1998.

John R Koza, Forrest H Bennett III, & Oscar Stiffelman. *Genetic programming as a Darwinian invention machine*. Springer, 1999.

Yemin Lan, J Calvin Morrison, Ruth Hershberg, & Gail L Rosen. Pogo-dba database of pairwise-comparisons of genomes and conserved orthologous genes. *Nucleic acids research*, 42(D1):D625–D632, 2014.

Thomas K Landauer, Peter W Foltz, & Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

Thomas K Landauer, Danielle S McNamara, Simon Dennis, & Walter Kintsch. *Handbook of latent semantic analysis*. Psychology Press, 2013.

Ben Langmead, Cole Trapnell, Mihai Pop, Steven L Salzberg, *et al.* Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

Claudia Leacock & Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283, 1998.

Jonathan G Lees, David Lee, Romain A Studer, Natalie L Dawson, Ian Sillitoe, Sayoni Das, Corin Yeats, Benoit H Dessailly, Robert Rentzsch, & Christine A Orengo. Gene3d: Multi-domain annotations for protein sequence and comparative genome analysis. *Nucleic acids research*, 42(D1):D240–D245, 2014.

Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, páginas 24–26. ACM, 1986.

Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, página 707, 1966.

Weiyuan Li & Hua Xu. Text-based emotion classification using emotion cause extraction. *Expert Systems with Applications*, 41(4):1742–1749, 2014.

T Warren Liao. Two hybrid differential evolution algorithms for engineering design optimization. *Applied Soft Computing*, 10(4):1188–1199, 2010.

Yung Siang Liau, Kay Chen Tan, Jun Hu, Xin Qiu, & Sen Bong Gee. Machine learning enhanced multi-objective evolutionary algorithm based on decomposition. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013*, páginas 553–560. Springer, 2013.

Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, páginas 296–304, 1998.

Julie Yu-Chih Liu, Jeng-Her Alex Chen, Chiang-Tien Chiu, & Juo-Chiang Hsieh. An extension of gene expression programming with hybrid selection. In *Proceedings of the 2nd International Conference on Intelligent Technologies and Engineering Systems (ICITES2013)*, páginas 635–641. Springer, 2014a.

Wei Liu, Hui Song, Jane Jing Liang, Boyang Qu, & Alex Kai Qin. Neural network based on self-adaptive differential evolution for ultra-short-term power load forecasting. In *Intelligent Computing in Bioinformatics*, páginas 403–412. Springer, 2014b.

Xiaoying Liu, Yiming Zhou, & Ruoshi Zheng. Sentence similarity based on dynamic time warping. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, páginas 250–256. IEEE, 2007.

Yongchao Liu & Bertil Schmidt. Swaphi: Smith-waterman protein database search on xeon phi coprocessors. *arXiv preprint arXiv:1404.4152*, 2014.

Kevin Lund, Curt Burgess, & Ruth Ann Atchley. Semantic and associative priming in high-dimensional semantic space. In *Proceedings of the 17th annual conference of the Cognitive Science Society*, volume 17, páginas 660–665, 1995.

Nitin Madnani, Joel Tetreault, & Martin Chodorow. Re-examining machine translation metrics for paraphrase identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, páginas 182–190. Association for Computational Linguistics, 2012.

Christopher D Manning & Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

Rada Mihalcea, Courtney Corley, & Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, páginas 775–780, 2006.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Tristan Miller, Chris Biemann, Torsten Zesch, & Iryna Gurevych. Using distributional similarity for lexical expansion in knowledge-based word sense disambiguation. In *COLING*, páginas 1781–1796, 2012.

Jessica Minnier, Ming Yuan, Jun S Liu, & Tianxi Cai. Risk classification with an adaptive naive bayes kernel machine model. *Journal of the American Statistical Association*, (just-accepted):00–00, 2014.

Michael Mohler, Razvan C Bunescu, & Rada Mihalcea. Learning to grade short answer questions using semantic similarity measures and dependency graph alignments. In *ACL*, páginas 752–762, 2011.

Michael Mohler & Rada Mihalcea. Text-to-text semantic similarity for automatic short answer grading. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, páginas 567–575. Association for Computational Linguistics, 2009.

Antonio Moreno. *Aprendizaje automático*. Ediciones UPC, 1994.

Jesús Oliva, José Ignacio Serrano, María Dolores del Castillo, & Ángel Iglesias. Symss: A syntax-based measure for short-text semantic similarity. *Data & Knowledge Engineering*, 70(4):390–405, 2011.

Alok Ranjan Pal & Diganta Saha. An approach to automatic text summarization using wordnet. In *Advance Computing Conference (IACC), 2014 IEEE International*, páginas 1169–1173. IEEE, 2014.

Kishore Papineni, Salim Roukos, Todd Ward, & Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, páginas 311–318. Association for Computational Linguistics, 2002.

Riccardo Poli & John Koza. *Genetic Programming*. Springer, 2014.

Roy Rada, Hafedh Mili, Ellen Bicknell, & Maria Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

Paul Rendell. *Turing machine universality of the game of life*. PhD thesis, University of the West of England, 2014.

Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.

Sharadindu Roy. Genetic algorithm based approach to solve travelling salesman problem with one point crossover operator. *International Journal of Computers & Technology*, 10(3):1393–1400, 2013.

Fernando Sánchez-Vega, Esaú Villatoro-Tello, Manuel Montes-y Gómez, Luis Villaseñor-Pineda, & Paolo Rosso. Determining and characterizing the reused text for plagiarism detection. *Expert Systems With Applications*, 40(5):1804–1813, 2013.

Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, & Bojana Dalbelo Bašić. Takelab: Systems for measuring semantic text similarity. In *Proceedings of the Sixth International Workshop on Semantic Evaluation*, páginas 441–448. Association for Computational Linguistics, 2012.

Roger N Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function. i. *Psychometrika*, 27(2):125–140, 1962.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, & John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, páginas 223–231, 2006.

Matthew G Snover, Nitin Madnani, Bonnie Dorr, & Richard Schwartz. Ter-plus: paraphrase, semantic, and alignment enhancements to translation edit rate. *Machine Translation*, 23(2-3):117–127, 2009.

Wei Song, Jiu Zhen Liang, & Soon Cheol Park. Fuzzy control ga with a novel hybrid semantic similarity strategy for text clustering. *Information Sciences*, páginas 156–170, 2014.

Murat Soysal & Ece Guran Schmidt. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation*, 67 (6):451–467, 2010.

Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, & Murat Demirbas. Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, páginas 841–842. ACM, 2010.

Dan Ştefănescu, Rajendra Banjade, & Vasile Rus. A sentence similarity method based on chunking and information content. In *Computational Linguistics and Intelligent Text Processing*, páginas 442–453. Springer, 2014.

Zhan Su, Byung-Ryul Ahn, Ki-Yol Eom, Min-Koo Kang, Jin-Pyung Kim, & Moon-Kyun Kim. Plagiarism detection using the levenshtein distance and smith-waterman algorithm. In *Innovative Computing Information and Control, 2008. ICICIC'08. 3rd International Conference on*, páginas 569–569. IEEE, 2008.

Chellammal Surianarayanan, Gopinath Ganapathy, & Manikandan Sethunarayanan Ramasamy. A practical approach to enhancement of accuracy of similarity model using wordnet towards semantic service discovery. *Demand-Driven Web Services: Theory, Technologies, and Applications*, páginas 245–266, 2014.

Nenad Tomašev, Miloš Radovanović, Dunja Mladenić, & Mirjana Ivanović. Hubness-based fuzzy measures for high-dimensional k-nearest neighbor classification. *International Journal of Machine Learning and Cybernetics*, 5(3):445–458, 2014.

George Tsatsaronis, Iraklis Varlamis, & Michalis Vazirgiannis. Text relatedness based on a word thesaurus. *Journal of Artificial Intelligence Research*, 37(1):1–40, 2010.

Amos Tversky. Features of similarity. *Psychological review*, 84(4):327–352, 1977.

Amos Tversky & Itamar Gati. Similarity, separability, and the triangle inequality. *Psychological review*, 89(2):123–154, 1982.

Juan Antonio Prieto Velasco. A corpus-based approach to the multimodal analysis of specialized knowledge. *Language resources and evaluation*, 47(2):399–423, 2013.

Karin Verspoor & Kevin Bretonnel Cohen. Natural language processing. *Encyclopedia of Systems Biology*, páginas 1495–1498, 2013.

Kirill A Veselkov, Lisa K Vingara, Perrine Masson, Steven L Robinette, Elizabeth Want, Jia V Li, Richard H Barton, Claire Boursier-Neyret, Bernard Walther, Timothy M Ebbels, *et al.* Optimized preprocessing of ultra-performance liquid chromatography/mass spectrometry urinary metabolic profiles for improved information recovery. *Analytical chemistry*, 83(15):5864–5872, 2011.

Rodolfo Villamizar, Jhonatan Camacho, Yudelman Carrillo, & Leonardo Pirela. Automatic sintonization of som neural network using evolutionary algorithms: An application in the shm problem. In *Advance Trends in Soft Computing*, páginas 347–356. Springer, 2014.

Mohammed Abdul Wajeed & T Adilakshmi. Different similarity measures for text classification using knn. In *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, páginas 41–45. IEEE, 2011.

Mengqiu Wang & Daniel Cer. Stanford: probabilistic edit distance metrics for sts. In *Proceedings of the Sixth International Workshop on Semantic Evaluation*, páginas 648–654. Association for Computational Linguistics, 2012.

Dominic Widdows. Geometry and meaning. *AMC*, 10:12, 2004.

Lei Wu, Steven CH Hoi, & Nenghai Yu. Semantics-preserving bag-of-words models and applications. *Image Processing, IEEE Transactions on*, 19(7):1908–1920, 2010.

Zhibiao Wu & Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, páginas 133–138. Association for Computational Linguistics, 1994.

Wang Yu, Wang Ruibo, Jia Huichen, & Li Jihong. Blocked 3× 2 cross-validated t-test for comparing supervised classification learning algorithms. *Neural computation*, 26 (1):208–235, 2014.

Torsten Zesch & Iryna Gurevych. Wisdom of crowds versus wisdom of linguists–measuring the semantic relatedness of words. *Natural Language Engineering*, 16(1): 25–59, 2010.

Torsten Zesch, Omer Levy, Iryna Gurevych, & Ido Dagan. Ukp-biu: Similarity and entailment metrics for student response analysis. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013), in Conjunction with the Second Joint Conference on Lexical and Computational Semantics (*SEM 2013)*, páginas 285–289. Association for Computational Linguistics, 2013.

Wen Zhang, Taketoshi Yoshida, & Xijin Tang. A comparative study of tf* idf, lsi and multi-words for text classification. *Expert Systems with Applications*, 38(3):2758–2765, 2011.

Xin Zheng & Ai Ping Cai. The method of web image annotation classification automatic. *Advanced Materials Research*, 889:1323–1326, 2014.

Ling Zhu, Feng Yang, Shuo Yang, Jinghua Li, Lirong Jia, Tong Yu, Bo Gao, & Yan Dong. The construction of semantic network for traditional acupuncture knowledge. In *Frontier and Future Development of Information Technology in Medicine and Education*, páginas 2239–2245. Springer, 2014.

Manuel Zini, Marco Fabbri, Massimo Moneglia, & Alessandro Panunzi. Plagiarism detection through multilevel text comparison. In *Automated Production of Cross Media Content for Multi-Channel Distribution, 2006. AXMEDIS'06. Second International Conference on*, páginas 181–185. IEEE, 2006.

Sven Meyer Zu Eissen & Benno Stein. Intrinsic plagiarism detection. In *Advances in Information Retrieval*, páginas 565–569. Springer, 2006.