Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Proyecto de grado
Ingeniería en Computación

# Aprendizaje semisupervisado de rasgos de temporalidad en el léxico del español

por

Mathias Claassen
Pablo Grill

Tutora: Aiala Rosá

Febrero 2017

# Resumen

Este proyecto surge como una colaboración entre el Grupo de Procesamiento del Lenguaje Natural de la Facultad de Ingeniería y el Departamento de Teoría del Lenguaje y Lingüística General de la Facultad de Humanidades y Ciencias de la Educación (DTLLG) en el marco del proyecto de Redes Temporales. Dicho proyecto definió una red de expresiones temporales agrupándolas en base a 15 rasgos semánticos definidos.

Este proyecto se basa en la investigación, diseño e implementación de una herramienta que permite la asignación automática o semiautomática de rasgos temporales de la mencionada red a nuevas expresiones temporales. Para ello se dispone de un conjunto inicial de expresiones ya clasificadas (aproximadamente 100), la definición temporal asignada a cada expresión y un modelo vectorial de palabras (Word Embeddings) generado a partir de un corpus en español.

En este contexto, la red temporal se convierte en un problema de clasificación semántica sobre el idioma en español.

Considerando que se disponía de poco ejemplos ya clasificados, muchos de los enfoques tradicionales de clasificación automática no pudieron ser utilizados, siendo necesaria la investigación y definición, mediante la experimentación, de nuevos algoritmos de clasificación.

El resultado del proyecto es una serie de algoritmos semisupervisados que asistirán a los integrantes del DTLLG en la definición de la red temporal. Dichos algoritmos se encuentran incorporados a una herramienta web de forma que su ejecución sea sencilla y amigable.

**Palabras Claves**: Red Temporal, Modelos Vectoriales, Clasificación Automática, Clasificación Semántica.

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción

## 1.1. Introducción al Proyecto

El Departamento de Teoría del Lenguaje y Lingüística General de la Facultad de Humanidades y Ciencias de la Educación (en adelante DTLLG) se dedica a la investigación y elaboración de proyectos relacionados a diferentes temáticas del idioma español. En este contexto, muchos de los proyectos e investigaciones desarrollados por el DTLLG cubren problemas y temáticas que también son de interés al grupo de Procesamiento del Lenguaje Natural (PLN) del Instituto de Computación de la Facultad de Ingeniería. En este escenario, el DTLLG y el grupo de PLN han colaborado y trabajado en conjunto desde hace varios años en diferentes proyectos.

En el contexto anteriormente detallado, este proyecto surge como una colaboración entre el grupo de PLN y el DTLLG sobre el proyecto de *Red Temporal*. En dicho proyecto el DTLLG propuso una serie de categorías que modelan diferentes rasgos que pueden tener las expresiones temporales del idioma español. Estas categorías conforman una red temporal que permite relacionar las expresiones en función de atributos y propiedades pertinentes a la temporalidad. Esta red temporal presenta un enfoque innovador en el ámbito ya que se analizan rasgos que no son comunes en recursos de este tipo.

Los integrantes del DTLLG asignados al proyecto de la red temporal se encuentran en el

desarrollo de una lista de expresiones que son consideradas temporales. La elaboración de dicha lista se realiza de forma manual considerando las siguientes fuentes:

- Diccionario de la lengua española (2001)
- Nuevo diccionario de uruguayismos (1993)
- Diccionario del español del Uruguay (2011)

Esta lista se encuentra en elaboración y se irá modificando en el transcurso del proyecto ya que el concepto de "temporalidad" puede ir variando a medida que se profundice en el análisis de la red temporal.

En una segunda etapa del proyecto tienen planificado clasificar cada una de dichas expresiones temporales en función de los rasgos definidos en la red temporal. El propósito de este proyecto es la creación de un algoritmo de clasificación automático que los asista en la asignación de dichos rasgos.

Si analizamos el problema desde un punto de vista más genérico, estamos ante un problema de clasificación semántica de múltiples categorías (*multiclass*) ya que existen múltiples rasgos en la red temporal. Dentro de las clasificaciones multiclass hay un subconjunto que se conocen como *multilabel*. En los casos multilabel, la salida del algoritmo de clasificación no es una única categoría, sino un conjunto de las mismas. En este proyecto el problema planteado es multilabel ya que a cada expresión temporal se le pueden asignar varios rasgos.

Este tipo de problemas es bastante común en PLN ya que muchos de los problemas típicos del área pueden reformularse como un problema de clasificación. Por lo tanto existe una gran cantidad de investigaciones, algoritmos e implementaciones que se encargan de abordar problemas de clasificación.

Sin embargo, este problema en particular tiene algunas características que lo diferencian de los problemas típicos de clasificación, generando que muchos de los algoritmos ya desarrollados no puedan utilizarse por no contar con las precondiciones requeridas por los mismos.

Estas particularidades convierten al proyecto en interesante y atractivo ya que se requerirá abordar el problema de clasificación desde un enfoque diferente a los usuales.

Este proyecto se enfocará en la investigación, elaboración y experimentación de diferentes algoritmos de clasificación automática que puedan servir de apoyo al DTLLG en la tarea de asignación de rasgos en la red temporal.

## 1.2. Objetivos del proyecto

Los objetivos del proyecto pueden resumirse en los siguientes puntos:

- Resolver la asignación automática (o semiautomática) de nuevas palabras temporales a los diferentes rasgos de la red temporal.
- Investigar diferentes alternativas para la clasificación automática de nuevas palabras temporales a categorías.
- Estudiar diferentes aplicaciones para las cuales se ha trabajado con la representación vectorial de palabras.
- Experimentar con los modelos vectoriales en el contexto de la asignación de rasgos temporales.
- Analizar otros enfoques diferentes a la utilización de modelos vectoriales para enfrentar el problema de asignación de rasgos temporales.
- Desarrollo de una herramienta informática que facilite el trabajo de asignación y visualización de palabras temporales en la red temporal.

## 1.3. Resultados esperados

Los resultados esperados se resumen en los siguientes puntos:

- Definición de uno o varios algoritmos de clasificación automática que permitan asignar nuevas palabras temporales a los rasgos de la red temporal.
- Implementación en python de los algoritmos definidos anteriormente.
- Elaboración de un documento que compare los resultados obtenidos para los diferentes algoritmos.
- Herramienta de visualización web para ejecutar los algoritmos y comprobar sus resultados.

## 1.4. Estructura del informe

El resto del documento se estructura de la siguiente forma. En el capítulo 2 se describe el problema sobre el que trata el proyecto. Se define e introduce la Red Temporal, el concepto de Word Embeddings (Modelos Vectoriales de Palabras) y su utilización en el contexto del presente proyecto. En el capítulo 3 se presenta el marco teórico con los diferentes conceptos investigados. Al final de este capítulo también se mencionan algunos trabajos en los que se basó este proyecto.

En el capítulo 4 se presentan los experimentos que se realizaron a lo largo del proyecto junto a su implicancia en el mismo. El capítulo 5 enumera los resultados obtenidos en el proyecto. Dichos resultados se presentan de forma cuantitativa y cualitativa.

El capítulo 6 detalla algunos aspectos funcionales y técnicos de la herramienta web desarrollada como producto final del proyecto. Dicha herramienta será utilizada por los integrantes del DTLLG para interactuar con los diferentes algoritmos y de dicha forma clasificar las nuevas expresiones temporales.

Por último, el capítulo 7 se enfoca en las conclusiones del proyecto y el trabajo a futuro que surge de este proyecto.

El informe culmina con la bibliografía consultada y un anexo donde se profundizan algunos de los temas tratados en el informe.

# Capítulo 2

# Descripción del Problema

En este capítulo se definirá desde un punto de vista lingüístico y técnico la *Red Temporal* sobre la que se trabajará en el transcurso del proyecto. Se definirán los rasgos temporales y como se relaciona el problema de clasificación al área de PLN. Además se introducirá el concepto de Word Embeddings y su aplicación al proyecto.

## 2.1. Proyecto Red Temporal

Como mencionamos anteriormente, este proyecto surgió como una colaboración entre el grupo de PLN del Instituto de Computación de la Facultad de Ingeniería y el DTLLG en el marco del proyecto de redes temporales. Esto hace que el proyecto en cierto modo forme parte de un proyecto interdisciplinar de mayores dimensiones. Por lo tanto, antes de profundizar en el problema específico desde el punto de vista del PLN describiremos brevemente el mismo desde un punto de vista más lingüístico.

El proyecto de red temporal se enfoca fuertemente en la asignación de rasgos a *expresiones temporales*. Se define *expresión temporal* a todas aquellas unidades (palabras o frases) que contengan en alguno de sus significados un rasgo de temporalidad. Por ejemplo, pueden considerarse expresiones temporales las siguientes unidades *almorzar, anciano, vez, antes, como taponazo, ahora, de repente, inmediato, minuto, a las apuradas, permanecer, durante*

y *al toque*.

Es importante destacar que en la definición de expresión temporal se menciona "alguno de sus significados". Esto hace que una expresión pueda ser considerada como temporal aunque su acepción más habitual no lo sea.

En el contexto de este proyecto se parte de una red temporal con unos 15 rasgos (categorías) definidos. Cada uno de estos rasgos fue definido en base a criterios semánticos y cubre diferentes aspectos y características de la temporalidad en el idioma español. Estos rasgos temporales que se definieron no son los más habituales cuando nos referimos a temporalidad, convirtiendo esta red temporal en un enfoque innovador del tema para el área.

Los rasgos que componen esta red temporal fueron definidos por el DTLLG de la siguiente forma:

- **Anclaje**: Expresiones relativas que se anclan en alguna eventualidad. El concepto de eventualidad se aplica a procesos, eventos y a los resultados de estos. Por ejemplo: *hasta, desde, tan pronto como, ni bien*.
- **Deíctico**: Expresiones deícticas que remiten al momento de la enunciación. Ejemplo son: *hoy, ahora, de ahora en mas, entonces*.
- **Desplazamiento**: Expresiones relativas que designan desplazamientos en relación con un momento o tiempo de referencia. Ejemplos: *alargar, dilatar, adelantado, precoz, precipitarse*.
- **Duración**: Expresiones que designan propiedades exclusivamente temporales y las duraciones que se asocian a ellos. Por ejemplo: *sempiterno, duradero, continuar, durar, el curso del tiempo, por los siglos de los siglos*.
- **Fase**: Expresiones que designan entidades cuyos significados se definen en relación con fases temporales. Ejemplos son: *albores, comienzos, cesar, juventud, todavía*.
- **Frecuencia**: Expresiones que designan frecuencia de algún hecho o evento. Por ejemplo: *cotidiano, frecuente, intermitente, de vez en cuando*.
- **Individuo**: Expresiones que designan individuos definidos en relación con su ciclo vital. Ejemplos: *adolescente, cordero, osezno, ternera, cincuentón*.
- **+/- Delimitado**: Expresiones que designan extensiones temporales delimitadas, o bien, no delimitadas, es decir, que poseen un inicio y un final o que no lo tienen.

Ejemplos del subconjunto **Más Delimitado** son: *fin de semana, siglo, infancia, racha, para largo*. Por otro lado, ejemplos de **Menos Delimitado** son: *eterno, infinito, nunca, jamás de los jamases, inmortal*.

- **+/- Especificado**: Expresiones que en conjunto designan extensiones caracterizadas por el hecho de que empiezan y culminan, es decir, delimitadas. Por lo tanto son una subcategoría de la categoría *Más delimitado*. Se distinguen dos subconjuntos en relación con la información que hace referencia a los límites. El subconjunto **Más Especificado** contiene piezas cuyos significados contienen una información precisa acerca del punto de inicio y de culminación, ambos intrínsecamente establecidos. Ejemplos de este subconjunto son: *mes semestre, fin de semana, Renacimiento, feriado.* Por otro lado, el subconjunto **Menos Especificado** contiene piezas cuyos significados carecen de esta información. Ejemplos son: *pasajero, transitorio, rato, temporada, a fines de, ocaso*.
- **+/- Recurrente**: Expresiones que designan estadios recurrentes o no recurrentes. Expresiones **Más Recurrentes** se repiten cíclicamente, como por ejemplo estaciones, días de la semana o nombres de eventualidades culturales. Ejemplos: *Semana de Turismo, verano, mañana, abril, viernes.* Por otro lado, las expresiones **Menos Recurrentes** no se repiten. Ejemplos son:*infancia, Neolítico, clásico, pasado*.
- **Orden**: Expresiones que se caracterizan porque sus denotados mantienen relaciones de orden. Ejemplos: *Edad Media, agosto, siguiente, viernes*.
- **Puntual**: Expresiones que designan puntos temporales para los que idealmente su comienzo y final coinciden en el tiempo. Por ejemplo: *instante, medianoche, repentino, estirar la pata, nacimiento*.
- **Tiempo- Espacio**: Expresiones que designan relaciones espacio-temporales. Ejemplo: *aceleración, lento, celeridad, como bala*.
- **Tiempo-Manera**: Expresiones en las que coexisten informaciones de tiempo y de manera. Por ejemplo: *inesperadamente, lentamente, precipitadamente, raudo, a todo trapo*.
- **Transformatividad**: Expresiones que denotan cambios de estado o cambios en un proceso. Ejemplos son: *nacer, finalizar, envejecer, volverse, evolución*.

Para cada uno de estos rasgos se parte inicialmente con unas 10 a 15 palabras de ejemplo.

Cada expresión temporal puede pertenecer a varios rasgos de la red. Por ejemplo, si

analizamos la expresión temporal *Abril* (cuarto mes del año) vemos que la misma pertenece a los rasgos *+/-Recurrente*, *Orden*, *+/-Delimitado* y *+/-Especificado*.

Se pretende que al finalizar este proyecto se disponga de una red de palabras que permita dos posibles visualizaciones:

- Visualización desde el punto de vista de palabras. Dada una palabra es de interés conocer qué rasgos de los definidos en la red presenta la misma.
- Visualización desde el punto de vista del rasgo. Dado un rasgo temporal es de interés conocer qué palabras tienen ese rasgo asociado.

Una vez que se disponga de la red temporal se tendrá un insumo que permitirá profundizar en el tema de temporalidad.

## 2.2. Proyecto Red Temporal desde PLN

Dado el contexto de la red temporal definida por el DTLLG, se detecta que el grupo de PLN de la Facultad de Ingeniería puede participar dando apoyo en la etapa de asignación de rasgos temporales a las diferentes expresiones temporales. Se puede investigar e implementar un algoritmo automático de asignación de rasgos semisupervisado. De esta forma se facilita el trabajo de asignación de rasgos a los integrantes del DTLLG, agilizando el avance del proyecto. Por otro lado, la creación de este algoritmo permitirá investigar el abordaje desde el punto de vista del PLN a un problema nuevo, generando de esta forma una ganancia para ambas partes.

Considerando lo anterior, en este proyecto se investigará la etapa de asignación de rasgos, dejando la definición de expresiones temporales a los integrantes del DTLLG.

Como fue mencionado anteriormente, nos encontramos ante un problema de clasificación semántica múltiple categoría (multiclass). Este caso de clasificación cuenta con las siguientes particularidades:

- *Bajo volumen de datos ya clasificados*. Se dispone de aproximadamente 15 ejemplos por cada categoría. Esto dificulta el abordaje del problema mediante métodos de

clasificación supervisados ya que no se dispone del volumen de datos suficiente para entrenar los mismos.

- *Similitud de categorías o límites entre categorías poco claros*: Algunas de las categorías que se definen son difíciles de diferenciar para personas ajenas al DTLLG. Es de esperar que dichas categorías sean difíciles de diferenciar de forma automática por el algoritmo desarrollado. Sumado a esto, hay categorías que son subconjuntos de otras categorías.
- *Definición de categorías muy específicas:* Algunas de las categorías definidas son tan específicas que determinar si una determinada expresión temporal debe pertenecer a la misma genera discrepancias entre los propios integrantes del DTLLG.
- *Conjunto de expresiones temporales en construcción*: El conjunto de expresiones con el que se trabajó en el proyecto fue evolucionando en forma paralela al mismo. Esto ocasiona que a medida que se avanza en la investigación vayan cambiando los insumos y puedan surgir situaciones imprevistas.
- *Conjunto de categorías en construcción*: Así como el conjunto de expresiones temporales no estaba completo inicialmente, también la definición de las distintas categorías no estaba en una versión final. A lo largo del proyecto se trabajó con las categorías definidas inicialmente pero en algunas reuniones surgió por parte del DTLLG dudas de si la definición de categorías es la más adecuada.
- *Red Temporal innovadora:* El estar ante un enfoque innovador provoca que no se disponga de trabajos anteriores que hayan analizado el mismo. Se deberá partir de proyectos e investigaciones sobre problemas similares y adaptar dicho conocimiento a este contexto en particular.

Considerando estas características el abordaje del problema se presenta difícil en una primera instancia. El mecanismo de trabajo que se desarrolló fue basado fuertemente en la experimentación, desarrollando la misma en varias iteraciones. En cada iteración se ponían a prueba hipótesis de trabajo que estaban fundadas en la bibliografía consultada, los experimentos realizados anteriormente o simplemente una idea que parecía intuitiva. De esta forma se fue avanzando en la generación de un algoritmo de clasificación que cubre las expectativas del DTLLG.

Finalizado el proyecto se obtuvo como resultado una herramienta web con el objetivo de facilitar la tarea de asignación de rasgos a las expresiones a los integrantes del DTLLG.

Esta herramienta tiene el propósito de facilitar en el futuro la ejecución de los algoritmos de clasificación por parte de los integrantes del DTLLG de forma que puedan clasificar nuevas expresiones temporales.

## 2.3. Utilización de Modelos Vectoriales de Palabras

Considerando que se dispone de poca información en la definición de las categorías de la red temporal, se propuso en una etapa muy temprana del proyecto la utilización de modelos vectoriales de palabras.

Los modelos vectoriales permiten agregar, por cada palabra o frase, información semántica de la misma en una forma interpretable para la computadora como lo son los vectores de números. Esto es un insumo muy importante en un contexto de clasificación semántica.

Además, los modelos vectoriales de palabras se están utilizando mucho en la actualidad para resolver diversos problemas del PLN; generando muy buenos resultados en algunas ocasiones. Por otro lado, el grupo de PLN tiene interés en avanzar en la utilización de estos modelos en sus investigaciones y proyectos. Por lo tanto, incluir un modelo vectorial en este proyecto puede ayudar a obtener mejores resultados en la clasificación y en paralelo comprender con mayor detalle formas y técnicas de utilización de los modelos vectoriales. Este último punto es importante ya que independientemente del resultado del algoritmo de asignación de rasgos se podrá incrementar la base de conocimientos que dispone el grupo de PLN sobre modelos vectoriales.

Por lo mencionado anteriormente, los modelos vectoriales son una parte fundamental del presente proyecto.

# Capítulo 3

# Marco Teórico y Trabajos Relacionados

En este capítulo resumimos la investigación teórica realizada en las primeras etapas del proyecto. Los tópicos sobre los que se hablará son los siguientes:

- Modelos Vectoriales de Palabras (Word Embeddings) y su utilización para problemas del área de PLN.
- Algoritmos de clasificación automáticos. Se tratará principalmente sobre algoritmos *no supervisados* y *semisupervisados*.
- Similitud de Lesk y su utilización para la determinación de sinónimos a partir de definiciones de palabras.

De todos estos tópicos se dará una visión general del tema, profundizando con mayor detalle en aquellos aspectos que fueron más importantes para el desarrollo del proyecto.

Al final de este capítulo se mencionan además algunos papers que fueron importantes para el proyecto e influenciaron en gran medida muchas de las decisiones que se tomaron en el transcurso del mismo.

## 3.1. Modelos Vectoriales de Palabras

### 3.1.1. Introducción

En esta sección presentaremos los modelos vectoriales y su utilización en el ámbito del PLN. Nos enfocaremos principalmente en los aspectos que consideramos más relevantes para el presente proyecto. El contenido de esta sección es una recopilación de varias fuentes de datos [19, 34, 11, 15, 18, 5, 25, 16].

Los modelos vectoriales de palabras o Word Embedding (en inglés) son un recurso que puede ser utilizado como insumo para la resolución de varios de los problemas del área de PLN. Estos modelos han ganado gran popularidad en la última década debido a los buenos resultados que se han alcanzado utilizando los mismos y sus propiedades.

Esta reciente popularidad ha generado que la investigación de este tema en el área de PLN sea de interés para la comunidad académica, lo que produce que constantemente surjan publicaciones con nuevas formas de generar modelos, nuevas formas de utilizar los modelos, comparaciones entre modelos, etc.

En líneas generales, un modelo vectorial consiste en la codificación de las palabras y/o frases de un idioma en un vector numérico de grandes dimensiones (500 o 1000). Esta codificación permite tener un mapeo de "(palabra, vector)" que permite identificar cada palabra con su correspondiente vector. Contar con este mapeo es importante ya que la gran mayoría de los algoritmos utilizados están pensados para trabajar con números (sobre todo las redes neuronales).

Sin embargo, a diferencia de otros posibles métodos de codificación de palabras (*simple enumeración* o *one-hot encoding* por ejemplo) los modelos vectoriales tienen algunas propiedades que los hacen interesantes:

- *Generación automática*: Los modelos vectoriales de palabras se generan usualmente a partir de un corpus de entrada utilizando un método de generación automática (Ver Figura 3-1).
  El poder generar el modelo de forma automática permite no enfocarse en eso y simplemente utilizar el mismo como un insumo más para el problema a resolver.

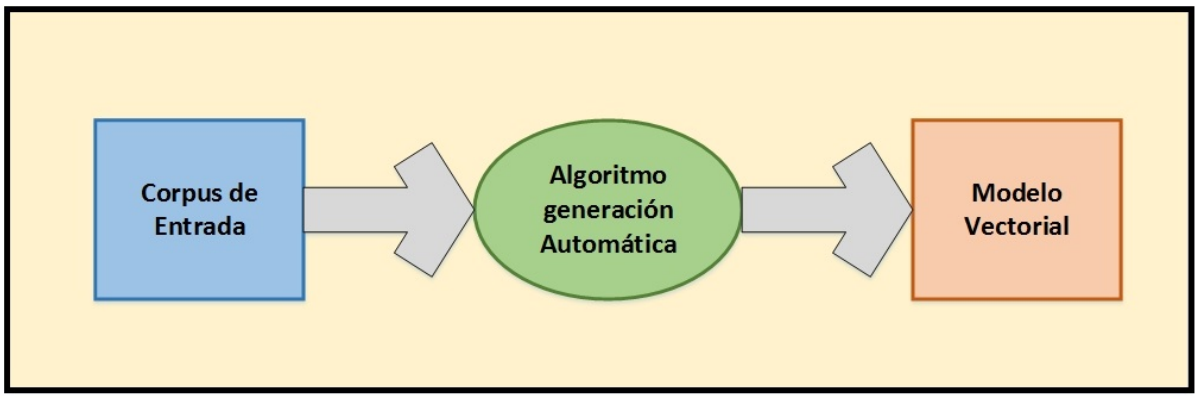


Figura 3-1: Generación de Modelo Vectorial

Existen varios algoritmos que permiten generar los modelos vectoriales entre los que se destacan los siguientes:

- ⋆ Latent Semantic Indexing (LSI)
- ⋆ Word2Vec
- ⋆ GloVe

- *Vectores densos de grandes dimensiones*: El modelar las palabras como vectores de más de 500 dimensiones es ideal para algoritmos basados en redes neuronales. Si disponemos de un conjunto de entrenamiento los suficientemente grande podemos utilizar los vectores de palabras para resolver varios de los problemas del área de PLN.
- *Contenido semántico en los vectores:* Los algoritmos de generación del modelo vectorial se basan en un corpus de entrada y la forma en que aparecen las palabras en el mismo. De esta forma, los vectores que se generan en el mismo tienden a contener condensados "todos los significados semánticos" de las palabras.
- "*Cercanía Semántica*": En estos modelos los vectores asociados a palabras "similares semánticamente" tienden a encontrarse "cerca" en el espacio vectorial.

A causa de estas propiedades y características los modelos vectoriales pueden ser utilizados para analizar palabras y las relaciones entre ellas. Se pueden definir distancias y funciones algebraicas en el modelo vectorial que pueden mapearse a relaciones gramaticales o semánticas entre las palabras asociadas a dichos vectores.

Por ejemplo, pueden definirse como sinónimos aquellas palabras cuyos vectores se encuen-

tren cerca dentro del espacio vectorial.

Otro ejemplo es la utilización de operaciones matemáticas entre los vectores de forma de obtener relaciones entre ellas.

Por ejemplo, la diferencia entre los vectores asociados a 'reina' y 'rey' es muy parecida a la diferencia entre 'mujer' y 'hombre'. Esto permite definir la igualdad expresada en 3.1:

$$reina - mujer = rey - hombre \tag{3.1}$$

o su equivalente 3.2:

$$rey = reina - mujer + hombre \tag{3.2}$$

Como los modelos vectoriales tienden a mantener las relaciones se puede utilizar la operación anterior para obtener nueva información.

Por ejemplo, sabiendo que la capital de Uruguay es Montevideo; queriendo saber cual es la capital de Bangladesh; podemos obtener esa respuesta con la ecuación 3.3:

$$CapitalDeBangladesh = Montevideo - Uruguay + Bangladesh \tag{3.3}$$

También podemos extraer información interesante y mucho más subjetiva de un modelo vectorial. Por ejemplo:

**amor** es a **indiferencia** como **miedo** es a [**apatía**, **temor**, **desconfianza**, **pasividad**]

Estos ejemplos utilizan el modelo vectorial como un todo, definiendo operaciones y distancias en el mismo. Para seguir este enfoque es necesario previamente detectar qué funciones se deben utilizar, lo cual requiere una etapa de análisis previa a la utilización.

Otra forma de utilizar los modelos vectoriales que no requiere esta etapa previa de análisis es incluir el vector asociado a cada palabra como un insumo de una red neuronal. Si bien al utilizar esta forma se evita la etapa previa de análisis; se requiere de un conjunto de datos lo suficientemente grande que permita entrenar la red neuronal.

A causa de las características que tienen estos modelos se pueden encontrar ejemplos de su utilización para las siguientes tareas:

- Análisis de Sentimiento
- Clasificación semántica no-supervisada
- Generación de modelos de lenguaje
- Word Clustering
- Analogy Questions

Respecto a los diferentes métodos de generación de modelos, en los últimos años han prosperado por sobre los demás: Word2Vec y GloVe.

### 3.1.2. Similitud y Distancia Coseno

Una de las formas de trabajar con modelos vectoriales es definir una distancia o función entre vectores que represente una relación semántica entre las palabras asociadas a los mismos.

En un espacio vectorial pueden definirse varias distancias, cada una de ellas con sus propiedades y características. Cuando hablamos de modelos vectoriales de palabras se aprecia que la distancia más utilizada es la similitud coseno.

Matemáticamente, la distancia coseno entre dos vectores $A$ y $B$ se define mediante la ecuación 3.4:

$$cosine\_distance(A, B) = \frac{\langle A, B \rangle}{\|A\| * \|B\|} \tag{3.4}$$

A partir de la definición de distancia coseno se define la similitud coseno con la fórmula 3.5:

$$cosine\_similarity(A, B) = 1 - cosine\_distance(A, B) \tag{3.5}$$

Al aplicar la similitud coseno en un espacio vectorial de palabras se aprecia que la misma

refleja de buena manera la equivalencia semántica de dichas palabras. Es decir, aquellas palabras cuya similitud coseno sea mayor serán más similares que aquellas palabras cuya similitud coseno sea menor.

La distancia coseno toma en cuenta solamente el ángulo entre los vectores involucrados y no la norma de los mismos. Esto puede resultar interesante cuando se trabaja con vectores no normalizados, o sea que no tienen norma 1.

### 3.1.3. Distancia euclidiana

La distancia euclidiana se usa mucho en álgebra y se define con la ecuación 3.6:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 \times (y_2 - y_1)^2} \tag{3.6}$$

La diferencia que tiene esta distancia con la distancia coseno es que toma en cuenta la norma de los vectores. Por esta razón, dos vectores que van en la misma dirección pero no tienen la misma longitud tendrán una distancia mayor a 0 mientras que la distancia coseno entre los dos sería igual a 0.

En el contexto de los modelos vectoriales de palabras muchas veces la norma de los vectores está asociada a cuántas veces ocurre la expresión representada en el corpus. Es por esto que normalmente se usa la distancia coseno ya que es deseado que palabras o expresiones similares tengan muy poca distancia entre sí sin importar la frecuencia con la que aparecen en el corpus.

Cabe destacar que la distancia euclidiana y la distancia coseno son equivalentes si se trabaja con vectores normalizados como es el caso de la mayor parte de este proyecto.

### 3.1.4. Word2Vec

Word2Vec fue presentado en 2013 por Mikolov et. al. [19].

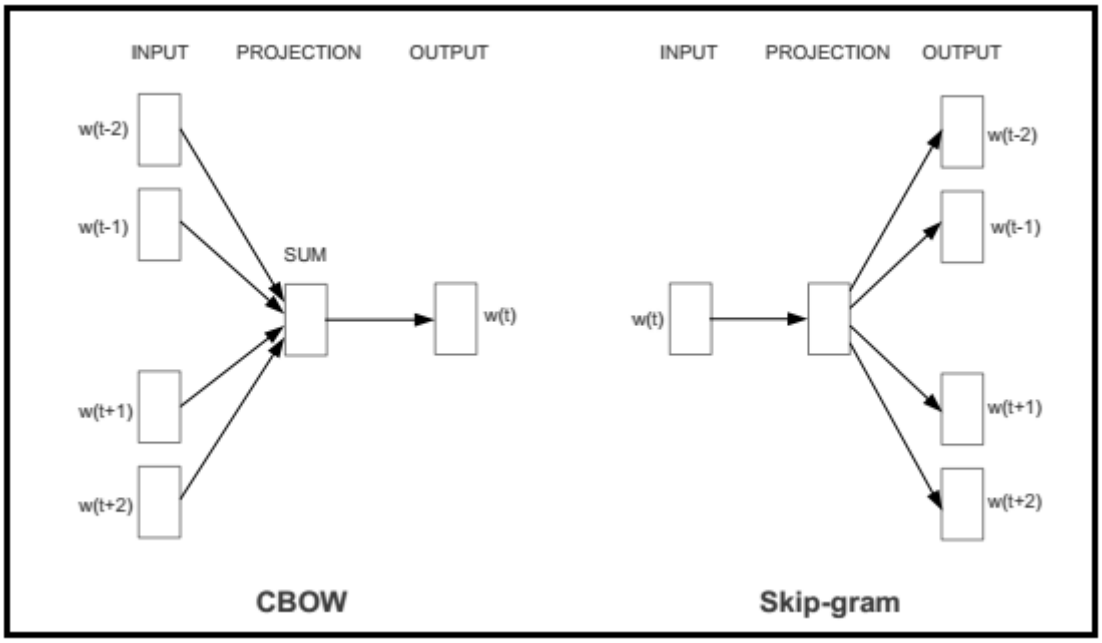


Figura 3-2: Comparación CBOW - Skip Gram[19]

A nivel de arquitectura, Word2Vec usa una red neuronal de dos capas para aprender los vectores asociados a cada palabra.

La idea principal detrás del algoritmo consiste en analizar los contextos en los que aparece una palabra, tomando como premisa que palabras similares aparecerán en contextos similares.

Esta idea de analizar los contextos se puede realizar mediante dos algoritmos posibles:

- Continuous bag of words (en adelante CBOW)
- Skip-Gram

CBOW toma un contexto y trata de predecir la palabra que va en ese contexto. Por otro lado, Skip-Gram realiza el análisis de forma inversa, tomando una palabra y tratando de predecir el contexto en el que va la palabra.

Esta diferencia entre los algoritmos puede visualizarse en la Figura 3-2.

Comparando ambos algoritmos, Mikolov menciona que Skip-Gram es el algoritmo que

retorna mejores resultados especialmente en las relaciones semánticas [19].

Si bien ambos métodos son de generación automática, tienen una serie de parámetros que permiten al momento de generar el modelo variar el mismo, como por ejemplo el tamaño de la ventana del contexto. Si esta ventana es demasiado chica entonces se perderá información de otras palabras relevantes al contexto, mientras una ventana demasiado grande puede aumentar el ruido que hay en el contexto. Al seleccionar el tamaño de la ventana hay que considerar que Word2Vec le da menos importancia a las palabras más alejadas, ponderando con mayor importancia las palabras más cercanas a la palabra estudiada.

Skip-gram en particular tiene dos formas de calcular las palabras candidatas a estar en el contexto de una palabra dada: "hierarchical softmax" y "negative sampling". Un estudio realizado por Levy y Goldberg asegura que "negative sampling" retorna los mejores resultados [16].

Otra optimización que realiza este algoritmo es que ofrece la posibilidad de hacer *sub-sampling*, o sea ignorar algunas de las palabras más frecuentes del corpus. Esto hace que no se consideren muchos de los *stopwords* y las palabras que entran en la ventana del contexto estudiado para cada palabra se ocupará con palabras más significativas.

La definición de los parámetros que se utilizan en la generación del modelo dependerá del problema a analizar y de las recomendaciones que existan en la bibliografía consultada.

### 3.1.5. GloVe

Otro algoritmo para obtener Word Embeddings es GloVe (Global Vectors) [25].

A diferencia de Word2Vec, GloVe es un método estadístico ya que se basa en el conteo de las co-ocurrencias de palabras. El método luego formula un problema de mínimos cuadrados para llegar a su resultado.

La idea principal del algoritmo se basa en una matriz de coocurrencias que tiene la estructura mostrada en la Figura 3-3.

Dicha matriz se construye contando las palabras que se encuentran en el contexto de la

| Probability and Ratio | $k = solid$ | $k = gas$ | $k = water$ | $k = fashion$ |
|---|---|---|---|---|
| $P(k\|ice)$ | $1.9 \times 10^{-4}$ | $6.6 \times 10^{-5}$ | $3.0 \times 10^{-3}$ | $1.7 \times 10^{-5}$ |
| $P(k\|steam)$ | $2.2 \times 10^{-5}$ | $7.8 \times 10^{-4}$ | $2.2 \times 10^{-3}$ | $1.8 \times 10^{-5}$ |
| $P(k\|ice)/P(k\|steam)$ | 8.9 | $8.5 \times 10^{-2}$ | 1.36 | 0.96 |

Figura 3-3: Matriz de coocurrencias[25]

palabra seleccionada.

Posteriormente se calcula la relación entre probabilidades de una palabra dadas otras dos que se quieren relacionar.

Si analizamos la Figura 3-3 podemos sacar las siguientes relaciones de palabras:

- Si la palabra k está relacionada a “ice” pero no a “steam”, en el ejemplo k =“solid”, la relación 3.7:
$$\frac{p(solid|ice)}{p(solid|steam)} \tag{3.7}$$
será grande (mucho mayor que 1).
- Si la palabra k está relacionada a “steam” pero no a “ice”, en el ejemplo k = “gas”, la relación 3.8:
$$\frac{p(gas|ice)}{p(gas|steam)} \tag{3.8}$$
será pequeña (mucho menor que 1).
- En cambio si la palabra $k$ no está relacionada con ninguna o si está relacionada con ambas de forma similar, la relación debería ser cercana a 1.

GloVe se presenta como un modelo que supera a Word2Vec en tests de analogías de palabras como las vistas anteriormente (rey es a reina como hombre a mujer) [25].

Sin embargo algunos análisis [26] demuestran que los resultados de ambos algoritmos son muy parecidos.

## 3.2. Clasificación, aprendizaje automático

### 3.2.1. Introducción

Considerando que el principal objetivo del proyecto es la clasificación automática de expresiones en la red temporal; en esta sección hablaremos sobre clasificación y aprendizaje automático. Se definirán e introducirán algunos de los conceptos más importantes del tema de forma de tener un contexto general. Además ser profundizará sobre algunos de los algoritmos investigados para el presente proyecto.

Por clasificación se entiende el proceso por el cual se trata de asociar ciertos objetos a un conjunto de clases ya definido. Esto puede hacerse mediante dos grandes métodos

- Construcción de reglas que determinen a qué categoría(s) pertenece cada objeto. Estas reglas deben definirse y programarse manualmente.
- Métodos de aprendizaje automático. Se entrenan modelos en base a conjuntos de datos ya clasificados. Se dividen en supervisados, semisupervisados y no supervisados en función de la intervención humana que se encuentra en el proceso de aprendizaje.

Normalmente usar un enfoque mixto es lo que brinda mejores resultados. Sin embargo, considerando que existen implementaciones de los métodos de aprendizaje automático que se pueden usar sin modificaciones; el costo de implementar el algoritmo mixto muchas veces se ve injustificado. El utilizar un algoritmo mixto requiere de intervención humana en el análisis del problema. Generalmente se utiliza un enfoque mixto cuando los resultados alcanzados mediante métodos de aprendizaje automático no cumplen las expectativas o no se disponen de los insumos suficientes para realizar el mismo (por ejemplo no se cuenta con suficientes datos ya clasificados para entrenar el algoritmo).

Los problemas de clasificación se pueden subdividir según las diferentes características que presentan.

Por ejemplo, los podemos clasificar según si hay una o varias clases a las que se les desea asociar objetos.

En el caso de que haya solo una clase entonces el resultado de la clasificación será un único valor (binario o booleano) indicando si un objeto pertenece o no a dicha clase y se

conoce como clasificación binaria.

En caso de que haya varias clases el resultado indicará a cual de estas clases pertenece el objeto. Este caso se conoce como "*multiclass*" [30] y podemos ver el problema como la situación de estar ante N casos de clasificación unitaria.

Cuando además de tener muchas clases, estas clases se solapan (el problema permite que los objetos sean asociados a varias clases) entonces se le llama "*multilabel*" [30]. En estos casos la cantidad de resultados posibles en teoría sería del cuadrado de la cantidad de clases.

En el caso de el presente proyecto, estamos ante un caso de clasificación *multilabel* ya que la red temporal cuenta con muchos rasgos y cada palabra puede pertenecer a más de uno.

Normalmente los resultados de las clasificaciones son probabilísticos, o sea que no son binarios sino que son números decimales entre 0 y 1 para cada clase. Esto permite que además de determinar a qué clase pertenece cada nuevo objeto se puede determinar con qué "seguridad" se hace esta asignación, o sea, se le asigna una probabilidad a la predicción.

Otra forma de clasificar los métodos de clasificación es por cuantos datos ya clasificados existen antes de empezar el proceso. Si hay una cantidad considerable de datos clasificados previamente entonces se habla de clasificación supervisada, donde el algoritmo de clasificación aprende en base a los ejemplos ya clasificados. Hay muchos métodos que implementan alguna forma de clasificación supervisada entre los cuales están las redes neuronales. Estos métodos por lo general requieren contar con una cantidad de ejemplos bastante mayor a la cantidad de valores de entrada (features) provistos para brindar buenos resultados.

Si no se cuenta con datos clasificados de antemano entonces se le llama clasificación no supervisada. En este tipo de clasificación se trata de agrupar objetos similares entre sí, sin disponer de mucha más información que los insumos del problema. En estos escenarios a menudo se usan algoritmos de clustering como por ejemplo K-Means [28].

Existe además un caso intermedio, llamado clasificación semisupervisada. En este caso existen algunos datos clasificados de antemano pero no son suficientes como para realizar clasificación supervisada. Los métodos de clasificación semisupervisada funcionan aplicando

el algoritmo en varias iteraciones. Entre iteración e iteración se encuentra la intervención humana que corrige algunos resultados del paso anterior. De esta forma el conjunto de datos clasificados correctamente va aumentando iteración a iteración hasta llegar a completar todo el dominio de entrada.

Existen algoritmos especialmente diseñados para problemas de clasificación semi supervisada. Entre ellos se destacan los algoritmos de Label Propagation y Label Spreading[29]. En estos casos se entrena con la parte de objetos ya asignados a una clase y otra parte de objetos aún sin clasificar. Ambos clasificadores empiezan por el conjunto de objetos ya clasificados y van agrandando ese conjunto iterativamente. Para ello construyen un grafo de similaridad entre todos los objetos a clasificar. La principal diferencia entre estos algoritmos es que Label Spreading modifica la matriz generada por el grafo de similaridad aplicando una función de costo para regularizar las entradas y es por lo tanto más robusto frente a ruido.

### 3.2.2. Métricas de Evaluación

Para analizar los resultados de una etapa de clasificación se comparan los resultados predichos por el algoritmo con los valores actuales. Se le llaman *falsos positivos* a aquellos objetos clasificados como positivos para cierta clase pero que en realidad no lo son. Así mismo se le llaman *falsos negativos* a los objetos clasificados como no pertenecientes a una clase pero que en realidad si pertenecen a la misma. A los objetos clasificados correctamente como positivos los denotaremos como *verdaderos positivos* así como los objetos correctamente clasificados como negativos se denominan *verdaderos negativos.*

Para evaluar un método o algoritmo se usan medidas como la *precisión*, el *recall* y la *medida F*. Estas medidas se calculan mediante las fórmulas 3.9, 3.10 y 3.11:

$$precision = \frac{verdaderos\_positivos}{verdaderos\_positivos + falsos\_positivos} \tag{3.9}$$

$$recall = \frac{verdaderos\_positivos}{verdaderos\_positivos + falsos\_negativos} \tag{3.10}$$

$$F = \frac{2 \times precision \times recall}{precision + recall} \tag{3.11}$$

Esto significa que intuitivamente el recall mide cuantos objetos positivos se encontraron y la precisión mide el porcentaje de positivos acertados de entre todos los objetos clasificados como positivos.

Asimismo la medida F trata de ser una medida global combinando las otras dos y penalizando fuertemente el resultado si alguno (precisión o recall) es muy bajo.

### 3.2.3. Active Learning

Active learning [1] es un caso especial de aprendizaje semisupervisado en el cual el algoritmo interactúa con el usuario para obtener datos clasificados. Este método se aplica cuando se tienen muchos datos no clasificados y el costo de clasificación manual es alto.

La idea principal de Active Learning es necesitar menos ejemplos clasificados para realizar la clasificación. Para esto, el algoritmo elige los ejemplos que el usuario va a clasificar, tratando de elegir ejemplos valiosos. Con esto, se pueden aprender conceptos con menos ejemplos que en clasificación supervisada.

El método usado para encontrar estos ejemplos a clasificar es un tema que se ha investigado mucho en el último tiempo generando varias propuestas. Por ejemplo, se pueden elegir los ejemplos para los cuales el algoritmo actual tenga más incertidumbre. Otro método es elegir aquellos ejemplos que cambiarían más al modelo.

Un problema de Active Learning normalmente comienza con unos pocos datos clasificados y una gran cantidad de ejemplos no clasificados. Se entrena un clasificador inicial sobre los ejemplos clasificados y luego se pasa a una fase de *selective sampling* en la cual se seleccionan algunos ejemplos para que el usuario los clasifique. Este proceso se repite iterativamente hasta llegar a converger.

### 3.2.4. Algoritmos de clustering

Un algoritmo de clustering o agrupamiento es un procedimiento que agrupa una serie de objetos en un cluster de acuerdo a un criterio. El objetivo es que los objetos en un cluster sean parecidos entre sí y no sean parecidos a objetos de otros clusters según el criterio elegido. Esos criterios son por lo general distancia o similitud. Estos algoritmos son usados por ejemplo en clasificación no supervisada porque no usan información de ejemplos clasificados previamente sino solamente es necesario definir qué distancia o similitud se utiliza.

Un algoritmo muy usado para hacer clustering es K-Means. K-Means crea una cantidad de clusters fijada de antemano y dada por la cantidad de centroides con los que se inicia el algoritmo. Se trata de que estos centroides estén inicialmente lejanos unos de otros para mejores resultados. K-Means asigna cada objeto al centroide más cercano formando así grupos de objetos. Luego calcula los nuevos centroides como los baricentros de los grupos formados en la etapa anterior y comienza a repetir estos dos procesos hasta que los centroides no cambien más.

Cuando se dispone de un problema en el que no hay muchos datos aparte del conjunto inicial de objetos a clasificar se suele utilizar clustering para hacerse una idea de la distribución de los objetos.

### 3.2.5. Redes Neuronales

Las redes neuronales artificiales son algoritmos de aprendizaje automático que deben su nombre a que guardan cierta semejanza con las redes neuronales cerebrales. Se destacan por tener varias capas de “neuronas” que ejecutan funciones tomando como entrada los resultados de las neuronas de la capa anterior multiplicados por un peso. Estos pesos se van ajustando para lograr el mejor resultado global posible. Para esto es necesario contar con un conjunto de ejemplos de entradas y salidas asociados para entrenar la red neuronal. Por lo tanto, las redes neuronales son algoritmos de clasificación supervisada.

La primer capa de las redes neuronales es llamada la capa de entrada mientras la última es la capa de salida. Todas las capas intermedias (si las hubiese) son llamadas capas

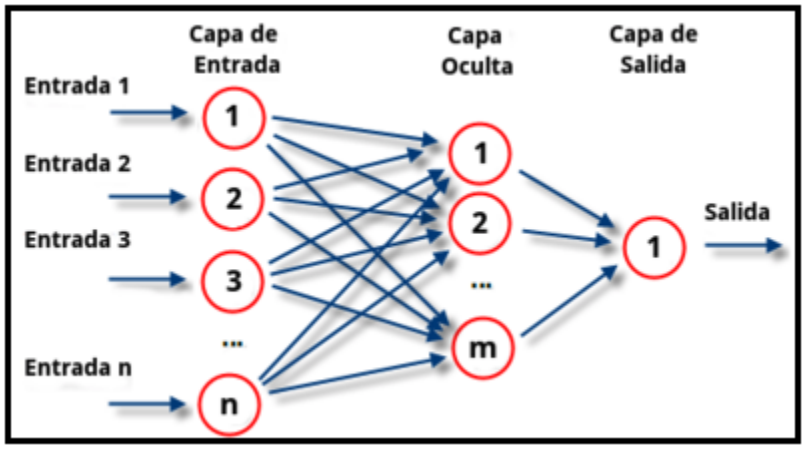


Figura 3-4: Arquitectura de Red Neuronal[21]

ocultas. Un ejemplo gráfico de una red neuronal se puede apreciar en la Figura 3-4.

Las redes neuronales son usadas en muchas áreas hoy en día. Uno de los casos de uso más comunes es el reconocimiento de patrones, como por ejemplo el reconocimiento facial o la digitalización de letras manuscritas.

Otra área donde se usan es para predecir valores futuros como el clima o valores de la bolsa.

### 3.2.6. Aprendizaje Automático en Red Temporal

En el marco de este proyecto, muchos de los métodos y algoritmos detallados anteriormente no podían ser aplicados a causa de la escasa cantidad de ejemplos ya clasificados que disponíamos. Esta característica hace que los métodos supervisados no puedan aplicarse directamente ya que no se dispone de suficientes datos para entrenarlos. Considerando este punto, el desarrollo del proyecto se baso fuertemente en la elaboración de algoritmos de clasificación semisupervisados.

La estrategia utilizada consistió en la elaboración de algoritmos basados en reglas que permitían con muy poca información sugerir candidatos para cada uno de los rasgos de la red temporal. Dichos candidatos son corregidos manualmente por integrantes del DTLLG. De esta forma, mediante la clasificación semisupervisada (en un enfoque similar al Active

Learning) se puede en algunas iteraciones expandir el conjunto de expresiones clasificadas rápidamente.

Sumado a esto, se definieron algoritmos supervisados que realizan clasificación automática sin intervención humana. Dichos algoritmos son LabelSpreading, LabelPropagation y una red neuronal. El propósito de estos algoritmos es que sean utilizados cuando se disponga de un conjunto de datos clasificados suficientemente grande para entrenar a los mismos.

De esta forma, mediante la ejecución iterativa de los diferentes algoritmos se podrán asignar rasgos para todas las expresiones temporales de la red.

## 3.3. Algoritmo y Similitud de Lesk

En el transcurso del proyecto se detecto la necesidad de detectar similitud de palabras en función de sus definiciones. Una de los algoritmos que permite automatizar esta tarea es el algoritmo de Lesk.

El algoritmo de Lesk fue publicado por Michael Lesk en 1986 [14] y se usa principalmente para desambiguar sentidos de palabras (Word Sense Disambiguation). Esto es cuando una palabra admite varios significados semánticos y se pretende determinar cuál de ellos es utilizado en un contexto específico.

Por ejemplo, tenemos la palabra “banco” que tiene los siguientes significados

- Empresa comercial que realiza operaciones financieras con el dinero procedente de accionistas y clientes.
- Asiento para varias personas, largo y estrecho; a veces está fijado al suelo.

Queremos determinar cuál de esos significados se usa en el siguiente texto:

“Los ladrones asaltaron el banco.”

Queremos que el algoritmo utilizado de desambiguación devuelva la primera definición ya que es la que aplica al texto citado.

Para esta tarea, el algoritmo de Lesk asume que palabras que aparecen cercanas a una

palabra en un texto comparten un tema común. Una versión simplificada de Lesk compara las palabras que rodean a la palabra objetivo en el texto. Posteriormente analiza todos los posibles significados de dicha palabra. Se tomará la definición que contenga mayor cantidad de palabras en común al contexto. Para esto se necesita de definiciones de algún diccionario. El algoritmo se podría definir así:

- Para cada significado de la palabra contar cuantas palabras ocurren en su definición y en el contexto (o cercanía) de la palabra a la vez.
- El significado elegido será el que tenga la cantidad de palabras repetidas más alta.

Si analizamos el concepto base del algoritmo fácilmente se aprecia que puede aplicarse para encontrar sinónimos (considerando como sinónimos aquellas palabras que compartan palabras en su definición). Para obtener una medida de similitud se puede usar la siguiente fórmula 3.12:

$$sym(s_1, s_2) = overlap(gloss(s_1), gloss(s_2)) \tag{3.12}$$

donde *gloss(w)* es la definición de diccionario de la palabra *w*. Fórmulas parecidas a esta están implementadas para WordNet [3, 36] en NLTK [4].

Cuando se utiliza el algoritmo de Lesk para la definición de sinónimos se denomina al mismo *Similitud de Lesk*.

## 3.4. Trabajos Relacionados

Al analizar el problema principal de este proyecto (asignación de rasgos temporales a expresiones) se detecta que no se encuentra en la bibliografía un proyecto que aborde dicha problemática. Esto se debe a que el enfoque abordado por el DTLLG es innovador desde un punto de vista lingüístico. Además, como ya mencionamos anteriormente, desde el punto de vista de PLN este problema de clasificación no es trivial ya que presenta características que dificultan su abordaje.

En este escenario se dificulta contar con proyectos anteriores que permitan definir el

estado del arte desde el punto de vista de las expresiones temporales. Sin embargo, existen numerosos proyectos y papers que resuelven problemas de clasificación semántica de similares características que sirvieron de base para este proyecto. En esta sección detallaremos algunos de los principales papers y cuál fue su aporte al proyecto.

### 3.4.1. Generación del Modelo Vectorial

Uno de los papers consultados sobre estrategias y metodologías de generación de modelos vectoriales fue *Improving Distributional Similarity with Lessons Learned from Word Embeddings*[16].

En dicho paper se comparan varios algoritmos de generación de modelos vectoriales respecto a Word2Vec. Los algoritmos que se analizan en este paper son:

- PPMI
- SVD factorization
- Word2Vec (SGNS - con Skip-Gram y negative sampling) [19]
- GloVe [25]

Dicha comparación se enfoca fuertemente en los hiperparámetros que permite Word2Vec y cómo su utilización afecta al modelo generado.

Dichos hiperparámetros son clasificados en 3 categorías:

- *Pre-processing hyperparameters*: Son parámetros que se aplican en los algoritmos de preprocesamiento previo a la generación del modelo vectorial. Estos hiperparámetros permiten afectar el corpus de entrada previo a la generación de vectores de forma de depurar ciertas características no deseables.
- *Association Metric Hyperparameters*: Son los parámetros que aplican sobre la propia generación del modelo vectorial. Estos parámetros permiten modificar la forma en que el algoritmo relaciona palabras con sus contextos, modificando de esta forma la generación de los vectores. Modificando estos parámetros se pueden generar modelos cuyos vectores reflejan diferentes relaciones semánticas/sintácticas.
- *Post-processing Hyperparameters*: Muchas veces el modelo generado mediante métodos automáticos no supervisados no contiene todas las cualidades que se requieren

para el problema deseado. En estos casos se suele refinar el modelo mediante métodos supervisados. En esta "refinación" del modelo aparecen estos hiperparámetros.

El resto del paper se enfoca en ver como estos hiperparámetros se pueden aplicar a los algoritmos analizados y qué resultados se pueden obtener.

Los experimentos se evaluaron mediante la aplicación de experimentos de *Word Similarity* y *Word Analogy*. En esta parte del paper no vamos a profundizar en este informe ya que no fue relevante para el proyecto.

El aporte de este paper al proyecto podría resumirse en los siguientes puntos:

- Entender los diferentes hiperparámetros que utiliza Word2Vec y como los mismos afectan al modelo generado. Este aspecto fue fundamental para las tareas de generación de modelo vectorial.
- Justificar la inclusión del modelo vectorial al proyecto. El hecho de que uno de los test realizados en el paper es el de *Word Similarity* aporta a justificar la teoría de que el modelo vectorial es un recurso que aporta valor al proyecto.
- Obtener una serie de consejos prácticos para el trabajo con modelos vectoriales de palabras.

### 3.4.2. Modelos Vectoriales para Clasificación Semántica

Al buscar ejemplos de problemas de clasificación semántica utilizando modelos vectoriales encontramos varios ejemplos con resultados prometedores.

Por ejemplo, en el paper *SINAI-EMMA: Vectores de Palabras para el Análisis de Opiniones en Twitter*[5] se utilizan vectores de palabras para realizar análisis de opiniones en twitter. El análisis de opiniones se basa fuertemente en el contenido semántico de una oración. Considerando que la red temporal basa la definición de sus rasgos mediante criterios semánticos, el encontrar un ejemplo de clasificación semántica utilizando modelos vectoriales es un hito importante.

El paper se basa fuertemente en la generación de un modelo vectorial orientado a la clasificación de opiniones según 6 niveles de polaridad *(P+, P, NEU, N, N+, NONE)*.

Para ello, propone la generación de un modelo vectorial a partir de métodos supervisados partiendo de recursos léxicos de opinión.

Los resultados obtenidos son muy buenos, logrando en algunos casos igualar o incluso superar el estado del arte en las áreas testeadas.

La información útil que aportó este paper al proyecto podría resumirse en los siguientes puntos:

- Se encontró un problema de características similares que puede resolverse utilizando modelos vectoriales de palabras. Si bien la polaridad de una opinión no es igual a la clasificación temporal de nuestro proyecto; ambos problemas radican en el contenido semántico de una expresión.
- Muestra la generación de un modelo vectorial en base a métodos supervisados. El utilizar métodos supervisados permite obtener vectores que modelan la opinión de buena manera. Sin embargo, para la utilización de métodos supervisados es necesario contar con un corpus de entrenamiento lo suficientemente grande.

Otro ejemplo de análisis de sentimiento con modelos vectoriales se presenta en el paper *Learning Word Vectors for Sentiment Analysis* [18].

Este paper es similar al anterior en el hecho de que se enfoca en utilizar un modelo vectorial para realizar tareas de análisis de sentimiento.

A diferencia del paper anterior, en el cual el modelo se construye directo de un corpus mediante métodos supervisados, en este paper se parte de un modelo vectorial generado de forma no supervisada y se refina el mismo de forma que se represente mejor la polaridad de las expresiones.

En este paper se afirma que los métodos automáticos de generación suelen generar buenos modelos vectoriales para las tareas como Word Sense disambiguation, Name Entity Recognition, PoS Tagging y Document Retrieval.

En el caso del análisis de sentimiento se menciona que los métodos automáticos tienden a perder información referente a la polaridad de la opinión, no reflejando la misma en el vector generado. A causa de esto, en el paper se mencionan algoritmos y estrategias de entrenamiento supervisado de forma de incluir en modelos vectoriales información

referente al sentimiento.

Lo que aportó este paper al proyecto fue el hecho de que la necesidad de "refinar" los modelos vectoriales es una particularidad del análisis de sentimientos. Existen problemas del área de PLN que pueden resolverse por los modelos generados mediante métodos automáticos no supervisados obteniendo buenos resultados. Esta información fue crucial en el proyecto ya que dado el escaso número de ejemplos clasificados con que contamos, la elaboración de un algoritmo supervisado para refinar un modelo vectorial no era viable.

### 3.4.3. Similitud Coseno en Modelos Vectoriales

Por último, el paper *Linguistic Regularities in Sparse and Explicit Word Representations*[15] fue muy importante para el desarrollo del proyecto debido a que los principios principales de algunos de los algoritmos surgieron del mismo.

El tema principal de este paper radica en la utilización de los modelos vectoriales de palabras para modelar la similitud de palabras.

Se basa fuertemente en una serie de demostraciones y experimentos que realizó Mikolov previamente que demostraban la usabilidad de un modelo vectorial para la temática de similitud de palabras. La idea principal detrás de estos experimentos es la de generar vectores para posteriormente detectar similitud entre los mismos. Este paper contiene como trabajo principal la demostración de que dicho método es análogo a encontrar un vector que maximiza la combinación lineal de tres pares de palabras similares.

Sin embargo, el aporte de este paper al proyecto no fue por el trabajo en sí mismo; sino el marco teórico que incluye.

En dicho marco teórico se explica detalladamente la utilización de modelos vectoriales de palabras para la detección de similitud entre palabras. Sobre todo, se define la distancia coseno como la norma que mejor conserva las propiedades de similitud de palabras. Considerando que la distancia coseno es parte fundamental en uno de los algoritmos desarrollados en este proyecto; la importancia de este paper es clara.

## 3.5. Aplicación al proyecto

Los tópicos tratados en el marco teórico fueron fundamentales en el presente proyecto ya que influyeron en gran medida en las decisiones tomadas a lo largo del mismo.

Los modelos vectoriales de palabras permitieron agregar información semántica, aporte fundamental considerando los escasos ejemplos clasificados con los que contábamos. La mayoría de los algoritmos de clasificación desarrollados se basan fuertemente en los vectores de palabras y la similitud coseno.

Respecto a clasificación automática, el resultado final del proyecto son una serie de algoritmos semisupervisados. De esta forma, iteración a iteración se puede incrementar el conjunto de ejemplos clasificados. La elaboración de estos algoritmos se realizó de forma iterativa basándonos fuertemente en la experimentación y la investigación teórica realizada. En ese proceso de experimentación se evaluaron algoritmos supervisados, no supervisados (clustering) y algoritmos basados en reglas.

Por último, la similitud de Lesk fue fundamental en el proyecto para incluir en los algoritmos de clasificación las definiciones de las expresiones temporales como feature. Las definiciones de las expresiones eran un insumo disponible que previo a la inclusión del algoritmos de Lesk no le estábamos sacando provecho. Agregar las definiciones de las palabras como feature de algunos algoritmos permitió obtener mejores resultados para algunas categorías.

# Capítulo 4

# Experimentos Realizados

## 4.1. Metodología de trabajo

Considerando las características del problema y la información con la que contábamos decidimos desarrollar una metodología de trabajo basada fuertemente en la experimentación.

Tomamos esta decisión ya que en la bibliografía consultada no encontramos un problema que cuente con todas las características que requerimos. Sin embargo, como se detalló en la sección de trabajos relacionados (Sección 3.4), encontramos algunos papers que aportaron ideas que podían llegar a ser útiles para este proyecto.

Los experimentos se desarrollan en diferentes etapas. Cada etapa tiene una serie de objetivos y se basa fuertemente en algunas hipótesis de trabajo. Dichas hipótesis se definen en base a la bibliografía consultada, los experimentos anteriores y en algunas ocasiones ideas que se pretenden validar o refutar.

Además, algunas de estas ideas surgen de temas que se vieron en el seminario de Deep Learning aplicado al PLN[12] al que asistimos como oyentes en paralelo a la realización de este proyecto.

De esta forma, mediante la experiencia adquirida en cada experimento pudimos hacer

madurar una serie de algoritmos de clasificación que cumplen en gran parte con lo esperado en los objetivos del proyecto.

En este capítulo del informe nos enfocaremos en detallar cada experimento desarrollado, cuál era su propósito y qué valor aporta al proyecto.

## 4.2. Utilización de modelos vectoriales

La mayoría de los experimentos desarrollados en este proyecto se basan fuertemente en la utilización de un modelo vectorial de palabras. Esta decisión fue realizada al comienzo del proyecto ya que consideramos que es un insumo útil a causa de sus características y propiedades (Ver sección 3.1).

En resumen, los aspectos que vemos positivos y útiles para el proyecto podemos resumirlos en los siguientes puntos:

- Los modelos vectoriales permiten tener una representación numérica de cada palabra. Contar con dicha representación facilita la clasificación mediante algoritmos automáticos.
- A diferencia de otras estrategias de representación numérica de palabras, en los word embeddings los vectores contienen y presentan propiedades interesantes. Se pueden definir funciones y métricas que permiten modelar relaciones entre los vectores. Dichas relaciones matemáticas entre vectores pueden mapearse a relaciones semánticas entre las palabras asociadas a los mismos.
- Los modelos vectoriales contienen información semántica. Esto sucede ya que los algoritmos de generación de estos vectores se basan en la forma en que se utilizan las palabras. Bajo la premisa de que palabras con semántica similar se utilizan de forma similar, podemos llegar a asumir con cierta seguridad que el modelo representa en alguna medida las relaciones semánticas entre las diferentes palabras del idioma. Este punto está mencionado en varios papers (Ver sección 3.4). Además, existen algunos conjuntos de tests que permiten "medir" de alguna manera la calidad del modelo (Por ejemplo el publicado en [6]).
- Existen casos de usos exitosos en varios problemas del PLN, obteniendo en ocasiones

los mejores resultados para el área.

Por los puntos anteriores, utilizar un modelo vectorial agrega valor al proyecto y permite obtener resultados más exactos.

Sumado a esto, si bien los modelos vectoriales de palabras están en el área de PLN desde hace tiempo, en los últimos años han ganado notoriedad a causa de los buenos resultados que han aportado a diferentes aplicaciones. Por lo tanto, al utilizar modelos vectoriales en un proyecto estamos aportando información y datos sobre el tema que se pueden utilizar en futuros proyectos del grupo de PLN.

## 4.3. Metodología de evaluación

Considerando las características de la red temporal, muchas de las métricas y técnicas de evaluación comunes del área de PLN no podían ser aplicadas.

Por ejemplo, considerando los escasos ejemplos categorizados que teníamos no podíamos separar un subconjunto como partición de testing ya que perderíamos mucha información para el desarrollo de los algoritmos.

Además, al no contar con partición de testing no se pueden utilizar algunas de las medidas de performance estándar (por ejemplo *recall*).

Por las características mencionadas se decidió medir la performance de las pruebas con las siguientes consideraciones:

- *Las métricas se analizan categoría a categoría (rasgo a rasgo)*: Se tomó esta decisión ya que las categorías son declaradas mediante definiciones independientes. Al realizar las mediciones en función de las categorías se puede llegar a detectar por ejemplo que algunos rasgos son más sencillos de predecir que otros.
- *Se medirá precisión de las expresiones predichas*: Cada algoritmo desarrollado predecirá para cada categoría un conjunto de expresiones temporales como candidatos a pertenecer a la misma. De ese conjunto de expresiones se determinarán junto a los integrantes del DTLLG cuales pertenecen a la categoría de forma de definir la precisión.

- *Se medirán los falsos negativos*: Para ello se aplicará el algoritmo a los propios integrantes de la categoría.
- *Se contarán el total de expresiones temporales predichas por cada algoritmo*: Ante la imposibilidad de medir el recall de los algoritmos, consideraremos el total de palabras que se predijeron como un indicador fundamental al momento de evaluar los mismos.

## 4.4. Experimentos desarrollados

En esta sección se detallan las diferentes etapas de experimentos que se desarrollaron a lo largo del proyecto. Los mismos se presentan de forma cronológica, de forma que se refleje la evolución del trabajo realizado.

### 4.4.1. Etapa 1

#### Contexto, insumos y objetivos

Etapa desarrollada al comienzo del proyecto, posterior a la presentación del mismo y la primer reunión con el DTLLG.

Fue una etapa fuertemente orientada a profundizar en la comprensión del problema y evaluar diferentes alternativas de abordaje al mismo (desde el punto de vista del PLN).

Se investigó mucho sobre el tema de clasificación semántica y modelos vectoriales de forma de contar con un sustento teórico que permitiera fundamentar las decisiones a tomar en el futuro.

Por otro lado, en esta etapa se comienza con la realización de experimentos que permiten evaluar la aplicación de los conceptos teóricos investigados al problema en particular.

Bajo este escenario, los objetivos propuestos para esta etapa se resumen en los siguientes puntos:

- Comprender con mayor profundidad y detalle las diferentes categorías de la red propuesta por el DTLLG.
- Evaluar de forma práctica la viabilidad de utilizar un modelo vectorial de palabras para el abordaje de este problema. Esto incluye los siguientes aspectos:
  - ⋆ Evaluar diferencias entre GloVe y Word2Vec desde el punto de vista teórico y práctico.
  - ⋆ Familiarizarse con la utilización de los modelos vectoriales (GloVe y Word2Vec).
  - ⋆ Investigar y analizar de qué forma aplicar el modelo vectorial al problema planteado.
  - ⋆ Investigar herramientas (bibliotecas) que faciliten la inclusión de los modelos vectoriales en algoritmos de clasificación automática.
- Realizar una versión básica de algoritmo que permita contar con rápidos resultados para analizar la viabilidad del proyecto.
- Comenzar la elaboración de una base de conocimientos sobre el problema que sirva de base para las siguientes etapas.

Desde el punto de vista de insumos, en esta etapa del proyecto contamos con lo siguiente:

- Definición funcional de las categorías (rasgos) de la red temporal con entre 10 a 15 palabras representantes de cada una de ellas.
- Un modelo vectorial del idioma español generado del corpus de la Wikipedia mediante el algoritmo de Word2Vec (obtenido del grupo de PLN de la Facultad de Ingeniería).
- Modelos vectoriales con dimensiones 25, 50, 100, 150 y 200 generados del corpus de la Wikipedia mediante el algoritmo GloVe (obtenido del grupo de PLN de la Facultad de Ingeniería).

### Trabajo realizado

La primer decisión que se tomó en esta etapa fue la forma de modelar las categorías del estilo "+/-" . Al analizar sus definiciones se puede apreciar que por más que refieren al mismo rasgo temporal representan conjuntos semánticamente diferentes. Por lo tanto, pueden considerarse como dos categorías diferentes (excluyentes entre sí).

Considerando lo anterior, las categorías *+/- Delimitado*, *+/- Especificado* y *+/- Recurrente*

se definen de la siguiente manera:

- Más Delimitado
- Menos Delimitado
- Más Especificado
- Menos Especificado
- Más Recurrente
- Menos Recurrente

El segundo punto que abordamos en esta etapa fue evaluar formas de utilizar el modelo vectorial para nuestro problema de clasificación. Según lo investigado y presentado en el marco teórico (Sección 3.1), hay varias formas de utilizar un modelo vectorial en el área de PLN.

Sin embargo, dadas las características de nuestro problema, muchos de los algoritmos presentados no podían ser aplicados en este proyecto. Una de las formas más comunes de utilizar los vectores de palabras es incluir los mismos como insumo de una red neuronal. Este enfoque es muy utilizado debido a que no requiere de realizar ningún análisis del problema, la tarea de "*feature extraction*" se omite dejando que el propio entrenamiento de la red sea el encargado de determinar qué componentes del vector son relevantes para el problema presentado.

Sin embargo, esta metodología no es aplicable a nuestro problema (al menos de forma directa) ya que para la elaboración de una red neuronal debemos contar con un conjunto importante de datos ya clasificados que sirva como insumo del entrenamiento de la misma. En esta etapa del proyecto contamos solamente entre 10 y 15 palabras por categoría, valor muy bajo para entrenar una red neuronal. Sin embargo, a medida que se fue profundizando en el proyecto el número de palabras clasificadas aumentó, permitiendo en etapas posteriores el abordaje con una red neuronal.

Descartando la red neuronal para esta etapa, decidimos utilizar el modelo vectorial de una forma global. Este enfoque consiste en utilizar distancias y otras operaciones entre vectores de palabras de forma de definir y detectar relaciones. Al analizar las propiedades de las diferentes distancias nos inclinamos por utilizar la similitud coseno (SC). Decidimos esto ya que hay información que avala que es buena aproximación a la "similitud semántica" como se detalla en [15].

Si consideramos que dos palabras que son "similares semánticamente" se clasifican de igual manera en la red temporal, podemos utilizar la similitud coseno para determinar similitud entre palabras y de esta forma clasificarlas. Esta idea es bastante intuitiva y parece prometedora en una primera instancia a causa del sustento teórico que tiene la similitud coseno como medida de similitud semántica.

Por lo tanto, esta hipótesis fue la primera que decidimos poner en evaluación de forma experimental. Para ello diseñamos un sencillo algoritmo que aplica dicha idea para clasificar de forma automática las palabras. Dicho algoritmo (*Algoritmo Vector SC - V1*) se compone de los siguientes pasos:

- Definir para cada categoría un *vector representante* de la misma.
- Definir un *umbral de SC* que define si una palabra pertenece a la categoría o no.
- Para cada una de las palabras a clasificar realizar el siguiente procedimiento:
  - ⋆ Calcular SC del vector asociado a la palabra con respecto al vector representante de la categoría.
  - ⋆ Si el valor de SC está dentro del umbral: la palabra pertenece a la categoría, en caso contrario no pertenece.

La implementación del *Algoritmo Vector SC - V1* requiere encontrar un único vector que representa de forma semántica la categoría. La investigación sobre modelos vectoriales y experimentos relacionados no reveló ningún método o estrategia para realizar esta tarea.

Es lógico e intuitivo pensar que dicho vector debe generarse en función de las palabras pertenecientes (referentes) a la categoría. En esta etapa decidimos utilizar el vector promedio aritmético ya que intuitivamente parece ser una opción viable.

La definición del umbral la hicimos de forma experimental. Para ello ejecutamos el algoritmo para cada categoría evaluando los propios representantes de la misma. Los resultados obtenidos se pueden visualizar en el Cuadro 4.1. Si analizamos dichos resultados, se aprecia que en la mayoría de las categorías el mayor número de palabras cae en el umbral de *mayor a 0.5*. Considerando estos resultados, el valor 0.5 parece ser un buen umbral.

Para evaluar este algoritmo realizamos una implementación del mismo en un notebook de Jupyter (Python). Algunas consideraciones de dicha implementación son las siguientes:

| **Categoría** | **Promedio Similitud Coseno** | **Cantidad palabras Categoría** | **Similitud Coseno >0.5** | **0.45 <Similitud Coseno <0.5** | **Similitud Coseno <0.45** |
|---|---|---|---|---|---|
| Deíctico | 0.5 | 25 | 15 | 5 | 5 |
| Anclaje | 0.45 | 15 | 9 | 3 | 3 |
| Desplazamiento | 0.5 | 10 | 7 | 3 | 0 |
| Tiempo-Espacio | 0.65 | 7 | 5 | 1 | 1 |
| +Recurrente | 0.60 | 10 | 8 | 1 | 1 |
| -Recurrente | 0.62 | 8 | 6 | 1 | 1 |
| Orden | 0.65 | 5 | 4 | 0 | 1 |
| +Delimitado | 0.55 | 20 | 11 | 4 | 5 |
| -Delimitado | 0.58 | 11 | 8 | 1 | 2 |
| +Especificado | 0.52 | 18 | 13 | 2 | 3 |
| -Especificado | 0.40 | 20 | 12 | 7 | 1 |
| Puntual | 0.52 | 17 | 13 | 1 | 3 |
| Transformatividad | 0.62 | 26 | 20 | 4 | 2 |
| Duración | 0.47 | 16 | 7 | 6 | 3 |
| Frecuencia | 0.41 | 21 | 12 | 0 | 9 |
| Tiempo-Manera | 0.55 | 15 | 11 | 0 | 4 |
| Individuo | 0.5 | 26 | 16 | 5 | 5 |
| Fase | 0.4 | 19 | 12 | 1 | 6 |

Cuadro 4.1: Resultados Etapa 1

- Se utilizó la biblioteca *gensim*[27] para integrar los modelos vectoriales a Python
- Para el cálculo del vector promedio se utilizó la función *mean* provista por *numpy*

Para probar dicho algoritmo utilizamos como universo de palabras temporales las brindadas en la presentación del proyecto y realizamos pruebas variando el modelo vectorial.

En paralelo hicimos una representación gráfica con los vectores de las palabras de cada categoría. Para obtener dicho gráfico utilizamos T-SNE [37, 38, 39] de forma de reducir la dimensión de los vectores a 2 y así poder representarlos en un plano. Si analizamos dicha gráfica (Figura 4-1) se puede apreciar que las categorías se encuentran solapadas en el espacio vectorial. Atribuimos dicho comportamiento a los siguientes factores:

- Para graficar los vectores se bajó la dimensión de 100 a 2. Eso provoca que se pierda información. Puede suceder que dos vectores que aparecen solapados en 2 dimensiones presenten diferencias si analizamos todas las dimensiones.
- El problema de clasificación es *multilabel*; por lo tanto una misma expresión temporal puede pertenecer a varias categorías. Ante este escenario, es probable que se solapen las categorías en algunos puntos ya que tienen miembros en común.

**Resultados obtenidos y conclusiones**

Los resultados obtenidos en esta instancia fueron analizados por nosotros sin intervención de integrantes del DTLLG. Los mismos fueron satisfactorios en varios aspectos y podrían resumirse en los siguientes puntos:

- La utilización de un modelo vectorial y la explotación del mismo mediante la similitud coseno prometen ser viables para este problema. Esta conclusión se ve justificada debido a los siguientes aspectos:
  - ⋆ Al crear el vector promedio de las categorías y evaluar con el algoritmo las palabras pertenecientes a la misma vimos que había pocas palabras a una distancia mayor a 0.5. A estas palabras que caen fuera del umbral definido pero pertenecen a la categoría los llamaremos *outliers* o ejemplos lejanos.
  - ⋆ Para todas las categorías el algoritmo predice nuevos integrantes de las mismas. Al analizar los mismos vimos que en los casos de las categorías *Tiempo-Espacio*, *+Recurrente*, *-Recurrente*, *Orden* y *+ Delimitado* el número de falsos positivos

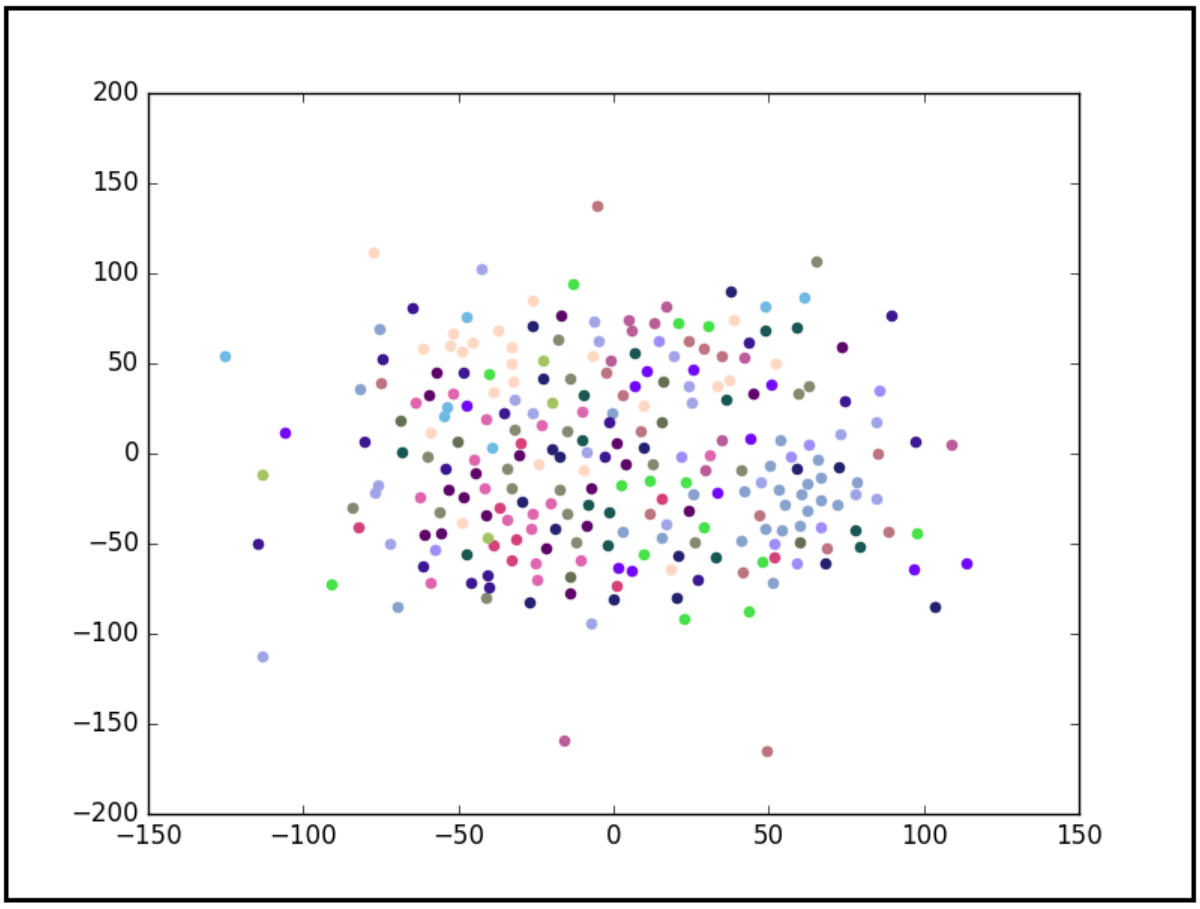


Figura 4-1: Representación Gráfica de categorías iniciales

| **Categoría** | **Cantidad Predicciones** | **Falsos Positivos** | **Falsos Positivos / Cantidad Predicciones** |
|---|---|---|---|
| Tiempo - Espacio | 3 | 0 | 0 |
| +Recurrente | 12 | 0 | 0 |
| -Recurrente | 3 | 0 | 0 |
| Orden | 3 | 0 | 0 |
| +Delimitado | 4 | 2 | 0.5 |

Cuadro 4.2: Resultados Predicciones Etapa 1

es bastante bajo como puede apreciarse en el cuadro 4.2. En las categorías restantes se predicen nuevas palabras pero el número de falsos positivos es mucho más grande.

- El algoritmo reveló que los verbos en infinitivo tienen una similitud coseno similar respecto a otros verbos en todos los modelos vectoriales con los que trabajamos. Eso provoca que para las categorías que tienen varios verbos, el algoritmo predice que la mayoría de los verbos restantes pertenecen a dicha categoría. Por ejemplo la categoría *Transformatividad* se modela con los siguientes integrantes: *nacer*, *morir*, *comenzar*, *finalizar*, *envejecer*, *madurar*, *rejuvenecer*, *revivir*, *resucitar*, *renacer*, *concluir*, *ultimar*, *caducar*, *expirar*, *extinguirse*, *convertir*, *convertirse*, *transformar*, *transformarse*, *volver*, *volverse*, *florecer*, *evolución*, *amanecer*, *crecimiento* y *menguar*. Como se puede apreciar, la mayoría de los representantes de esta categoría son verbos. La predicción que el algoritmo arroja son las siguientes palabras: *recordar : 0.634878559042, alargar : 0.646357180379, postergar : 0.655214578553, precipitarse : 0.722667612302, acelerar : 0.637612229274, correr : 0.661253631308, despertarse : 0.684323482504, extinguirse : 0.739046979203, desaparecer : 0.770658639452, continuar : 0.637961599699, permanecer : 0.690439671045, mantenerse : 0.696615999241, seguir : 0.653013378722, acostumbrar : 0.680849616003* y *acostumbrarse : 0.772810054788*. Las palabras predichas son todos verbos ya que en el espacio vectorial todos los verbos se encuentran "cerca" según la SC. Sin embargo, no todos los verbos temporales deben pertenecer a la categoría *Transformatividad*. Este resultado parece evidenciar que en los modelos vectoriales además de las relaciones semánticas se pueden deducir relaciones gramaticales y morfológicas.
- Los modelos vectoriales con que trabajamos en esta instancia no cuentan con frases,

solamente con palabras. Considerando que algunas de las expresiones temporales a clasificar en la red temporal son frases, hay que analizar la forma de incluir las mismas en el algoritmo de clasificación.

- Al comparar los diversos modelos vectoriales con los que trabajamos vimos que los resultados eran similares. Se evidenció que en el caso de los modelos GloVe, al aumentar la dimensión se reducía un poco el número de falsos negativos. Sin embargo la diferencias encontradas son mínimas y los resultados del modelo de GloVe de dimensión 200 no superaron a los del modelo Word2Vec (dimensión 100). Por lo que decidimos dejar para las siguientes etapas del proyecto solamente el modelo vectorial de Word2Vec.
- Al graficar los vectores pertenecientes a las diferentes categorías vemos que los mismos se encuentran bastante cerca unos de otros. No se puede evidenciar una clara separación de categorías. Esto puede deberse a los siguientes motivos:
  - ⋆ Al bajar dimensiones para graficar se pierde información de los vectores
  - ⋆ El modelo vectorial generado engloba por cada palabra todos los posibles significados semánticos de las palabras. Puede suceder que no se están englobando los sentidos temporales que se requieren para esta red o que los mismos queden "ocultos" bajos usos semánticos más usuales de las palabras.
  - ⋆ Es esperable que las categorías de la red temporal que son similares o hasta son subconjuntos unas de otras se solapan dentro del modelo vectorial.

Con estos resultados dimos por concluida la etapa 1 con un balance positivo de la misma. Los objetivos principales de la etapa se cumplieron en su gran mayoría de forma satisfactoria en el tiempo estimado. Los resultados obtenidos hacen prometedor el abordaje del problema de clasificación desde el punto de vista de PLN.

### 4.4.2. Etapa 2

#### Contexto, insumos y objetivos

En esta etapa continuamos con la investigación teórica y experimentación mediante notebooks de Jupyter en Python. Los insumos para esta etapa fueron los mismos que en la etapa 1, con la particularidad de que solamente utilizamos el modelo vectorial de

Word2Vec por lo mencionado anteriormente.

Fue una etapa orientada a profundizar en el problema y avanzar en el desarrollo del algoritmo de clasificación.

En este contexto se definen los siguientes objetivos.

- Disminuir el número de *outliers* (ejemplos lejanos) del algoritmo utilizado en la etapa 1. Si la propias palabras que están modelando la categoría son mal clasificadas es de esperar que las predicciones no tengan una buena calidad.
- Generar una serie de resultados a enviar al DTLLG de forma de tener un primer análisis de los resultados obtenidos.
- Analizar estrategias para incluir las frases en el algoritmo dada la importancia que les da el DTLLG a expresiones temporales de dicho tipo.
- Analizar variaciones del algoritmo de forma de mitigar el problema de los verbos en infinitivo detectado en la etapa anterior.
- Investigar y diseñar otros algoritmos diferentes al utilizado en la etapa 1 (*Algoritmo Vector SC - V1*) de forma de evaluar otras posibilidades.

### Trabajo realizado

Lo primero que analizamos fue formas de reducir el número de *outliers* (ejemplos lejanos) del algoritmo de la etapa 1. Es lógico pensar que aquellos vectores de palabras que quedan registrados como lejanos no aportan información útil al vector promedio. Además, es probable que la información que brinden tienda a perjudicar el valor generado por las restantes palabras.

Considerando la apreciación anterior, parece una buena idea eliminar dichas palabras de la generación del vector promedio. Para evaluar esta idea se modifica el *Algoritmo Vector SC - V1* de la siguiente manera:

- Por cada categoría se define el vector promedio de la misma en función de las palabras pertenecientes a la misma.
- Para cada categoría se evalúa la SC contra el vector promedio de aquellas palabras pertenecientes a la categoría. En función del valor de dicha similitud se generan dos

categorías:

- ⋆ Vectores cercanos: Aquellos para los que la similitud coseno es superior a 0.4
- ⋆ Vectores lejanos: Aquellos para los que la similitud coseno es inferior a 0.4

El valor 0.4 se seleccionó ya que en las pruebas de la etapa anterior se definió experimentalmente que el valor 0.5 era un valor adecuado para definir la pertenencia a una categoría. Considerando eso decidimos agregar un umbral de 0.1 de tolerancia para considerar un vector como cercano.

- Para cada categoría se crean dos nuevos vectores promedio en función de los vectores cercanos y lejanos.
- Se ejecuta para el resto de las palabras la similitud coseno en base a estos dos vectores. Si superan el umbral de 0.5 con alguno de los vectores las palabras se consideran como una predicción del algoritmo para dicha categoría. En este caso, por cada predicción se indica a que vector esta relacionada la misma (cercanos o lejanos).

Con este cambio, buscamos por un lado encontrar nuevas palabras cercanas al "núcleo" de la categoría (*palabras cercanas*) de forma de obtener resultados más precisos (mejorar precisión del algoritmo).

Además, en caso que el algoritmo sugiera palabras correctas respecto a los vectores lejanos (*outliers* o *falsos negativos* en el *Algoritmo Vector SC - V1*) se estaría detectando que la categoría, desde el punto de vista vectorial, está dividida en dos subconjuntos.

A este algoritmo nos referiremos como *Algoritmo Vector SC - V2*.

En paralelo evaluamos un nuevo algoritmo que no utiliza el concepto del vector promedio. Este nuevo algoritmo fue denominado "*Distancia a N*" y su principal propósito era aprovechar las virtudes de la similitud coseno sin utilizar el vector promedio (concepto que no tenía un sustento teórico).

La estructura de este nuevo algoritmo es la siguiente:

- Por cada categoría se realizan los siguientes pasos:
  - ⋆ Por cada palabra ajena a la categoría se calcula la SC respecto a cada palabra integrante (representante) de la misma.
  - ⋆ En caso de que se encuentren más de N palabras de la categoría que están en

un umbral definido (parametrizable), se considera la palabra como predicción.

Como claramente se aprecia, este algoritmo no utiliza el concepto de vector promedio y solo se enfoca en la similitud coseno, concepto que si tiene un fundamento bibliográfico que lo sustenta. Además el algoritmo permite variaciones modificando los valores de N (cantidad necesaria de palabras) y el umbral de SC considerado.

En paralelo a estos experimentos también probamos la implementación de un algoritmo de clustering. La idea de las pruebas de clustering eran ver cómo se agrupan las palabras en función de los vectores de las mismas sin incluir análisis ni otra información. Además, los algoritmos de clustering logran por lo general formar grupos y clases que son especialmente útiles en clasificación no supervisada (Ver sección 3.2). Considerando la escasa información que teníamos de las categorías, el hecho de contar con un método no supervisado era una aproximación que tenía sentido abordar.

Se realizaron varios experimentos usando principalmente el método K-Means (Detallado en la sección 3.2.4). Los resultados obtenidos con estos algoritmos generaron grupos de palabras que tienen cierta similitud semántica. Sin embargo no pudimos observar que los grupos generados por el algoritmo se relacionaron de forma directa con las categorías definidas en la red temporal. Encontramos algunos casos de grupos generados que podían llegar a estar incluidos en alguna de las categorías, pero no fue nada directo.

Ante la imposibilidad de determinar de forma directa las categorías asociadas a un cluster y el hecho de que contamos con categorías definidas de antemano dejamos de investigar esta línea de trabajo.

Otro hecho que fundamenta esta decisión es que las palabras por lo general pertenecen a más de una categoría y los algoritmos de clustering asignan los objetos a un solo cluster.

### Resultados obtenidos y conclusiones

Luego de finalizada esta etapa de experimentos enviamos los resultados obtenidos por los algoritmos *Algoritmo Vector SC - V2* y *Distancia a N* al DTLLG para evaluarlos.

El consolidado del análisis realizado por el DTLLG puede visualizarse en los cuadros 4.3 y 4.4.

| **Categoría** | **Cantidad de Palabras Categoría** | **Cantidad de Predicciones** | **Falsos Positivos** | **Precisión** |
|---|---|---|---|---|
| Deíctico | 15 | 4 | 3 | 0.25 |
| Anclaje | 18 | 16 | 9 | 0.31 |
| Desplazamiento | 11 | 25 | 22 | 0.08 |
| Tiempo-Espacio | 9 | 1 | 0 | 1 |
| +Recurrente | 12 | 9 | 0 | 1 |
| -Recurrente | 8 | 3 | 0 | 1 |
| Orden | 6 | 0 | 0 | 0 |
| +Delimitado | 23 | 14 | 0 | 1 |
| -Delimitado | 14 | 3 | 3 | 0 |
| +Especificado | 20 | 11 | 4 | 0.64 |
| -Especificado | 22 | 9 | 5 | 0.45 |
| Puntual | 20 | 22 | 18 | 0.18 |
| Transformatividad | 26 | 25 | 25 | 0 |
| Duración | 19 | 23 | 23 | 0 |
| Frecuencia | 28 | 11 | 11 | 0 |
| Tiempo-Manera | 22 | 2 | 1 | 0.5 |
| Individuo | 25 | 3 | 2 | 0.33 |
| Fase | 21 | 3 | 0 | 1 |

Cuadro 4.3: Resultados Distancia a N

| **Categoría** | **Cantidad de Palabras Categoría** | **Cantidad de Predicciones** | **Falsos Positivos** | **Precisión** |
|---|---|---|---|---|
| Deíctico | 15 | 12 | 7 | 0.33 |
| Anclaje | 18 | 23 | 14 | 0.39 |
| Desplazamiento | 11 | 22 | 21 | 0.05 |
| Tiempo-Espacio | 9 | 7 | 4 | 0.43 |
| +Recurrente | 12 | 20 | 4 | 0.8 |
| -Recurrente | 8 | 14 | 5 | 0.64 |
| Orden | 6 | 15 | 0 | 1 |
| +Delimitado | 23 | 35 | 3 | 0.91 |
| -Delimitado | 14 | 16 | 14 | 0.13 |
| +Especificado | 20 | 29 | 6 | 0.79 |
| -Especificado | 22 | 16 | 8 | 0.5 |
| Puntual | 20 | 35 | 26 | 0.26 |
| Transformatividad | 26 | 30 | 29 | 0.04 |
| Duración | 19 | 46 | 42 | 0.08 |
| Frecuencia | 28 | 2 | 2 | 0 |
| Tiempo-Manera | 22 | 12 | 8 | 0.33 |
| Individuo | 25 | 4 | 4 | 0 |
| Fase | 21 | 5 | 2 | 0.6 |

Cuadro 4.4: Resultados Similitud Coseno

El análisis de los resultados y el valor que obtuvimos de esta etapa pueden resumirse en los siguientes puntos:

- En todas las categorías se predijo alguna nueva palabra perteneciente a la misma. Si bien el desempeño fue diferente en cada categoría, el hecho de que en todas se haya logrado predecir al menos una palabra perteneciente a la misma afirma la idea de que el camino que estamos siguiendo es el correcto.
- La generación de dos vectores por cada categoría (vectores cercano y lejano) redujo de forma considerable la cantidad de palabras lejanas al centro de la misma. Sin embargo, dicho valor aún es demasiado alto para un algoritmo de clasificación automática. El hecho de eliminar los vectores "lejanos" no redujo tanto como se esperaba la predicción de falsos positivos.
- Se detectaron algunas categorías que presentaron resultados más acertados que otras. Estas categorías son *Orden*, *Individuo*, *Más Recurrente*, *- Recurrente*, *+Delimitado*, *+Especificado*. Sin embargo, el número de falsos positivo es alto en la gran mayoría de las categorías.
- No se detectan grandes diferencias entre los algoritmos *Algoritmo Vector SC - V2* y *Distancia a N*. Si bien los valores de presión y la cantidad de predicciones difieren, las categorías que presentan mejores resultados son las mismas para ambos algoritmos.
- La utilización de la similitud coseno como parte fundamental de los algoritmos se fortalece ya que los resultados de algunas categorías así lo indican. Es de esperar que a medida que avance el proyecto y con nuevas ideas se obtengan resultados aún mejores.

Finalizada esta etapa de experimentos aumentó la viabilidad de la generación de un clasificador automático. Contar con resultados validados por el DTLLG y resultados prometedores en algunas de las categorías parece indicar que las decisiones tomadas hasta el momento son acertadas y que se debe seguir por dicho camino.

### 4.4.3. Etapa 3

#### Contexto, insumos y objetivos

Como la lista de palabras temporales continuaba en construcción decidimos iniciar una

etapa enfocada en analizar nuevas ideas y resolver algunos de los problemas que quedaron pendientes en las etapas anteriores. Esta etapa se caracterizó por la evaluación en paralelo de diferentes experimentos con el fin de validar o refutar nuevas hipótesis.

Considerando el resultado de la etapa anterior decidimos continuar con la estructura del algoritmo definida ya que parecía ser viable por los resultados obtenidos. Las líneas de experimentación que seguimos en esta etapa podrían resumirse en los siguientes puntos:

- Obtener un vector representante de la categoría diferente al "*mean vector*". Se buscó una forma de representar el vector que conservará de mejor manera la propiedad de similitud coseno.
- Enfrentar el problema de "cercanía" de los verbos en infinitivo conjugándolos. Se enfocó este problema desde dos puntos de vista:
  - ⋆ Conjugación manual de verbos
  - ⋆ Conjugación automática de verbos
- Para aquellas palabras que no pertenecen al modelo vectorial buscar un sinónimo que sí pertenezca. De esta forma se puede utilizar el vector asociado al sinónimo en los algoritmos *Algoritmo Vector SC - V1*, *Algoritmo Vector SC - V2* y *Distancia a N*. Para localizar los sinónimos se utilizó la herramienta WordNet [36].

### Trabajo realizado

Una de las primeras tareas que abordamos fue la de generar un nuevo vector representante de la categoría. Como mencionamos anteriormente, en la bibliografía consultada no encontramos ningún proyecto ni experimento que genere un vector de las características deseadas.

Se investigó en profundidad la API de *gensim*. Dicha API dispone de una gran variedad de funciones para trabajar con modelos vectoriales de palabras, por lo tanto es un buen punto de partida para esta tarea. Al analizar en detalle la API no encontramos ninguna funcionalidad que dado un conjunto de palabras devolviera un vector que de alguna forma las representara. Sin embargo, al analizar los fuentes de la biblioteca vimos que para la implementación de la función *most similar* [10] se utiliza un vector generado a partir de las palabras positivas y negativas. Dicho vector tiene la propiedad de que se

encuentra “cercano” según la distancia SC a las palabras positivas pero “lejano” a las palabras negativas.

Considerando que la función de *most similar* se utiliza para encontrar palabras similares a las recibidas con buenos resultados nos pareció que dicho vector podría ser una mejor representación que el vector promedio utilizado en las etapas anteriores. Por otro lado, la generación de este vector permite además la inclusión de palabras negativas; funcionalidad no soportada por el *mean vector*.

Por lo anteriormente mencionado optamos por utilizar el cálculo del vector que usaba *gensim* en nuestro algoritmo de clasificación en lugar del promedio aritmético utilizado anteriormente.

Probamos con las categorías de la red temporal este nuevo algoritmo de generación del vector para evaluar su desempeño. En el caso de las categorías del estilo “+/-” optamos por colocar como palabras negativas aquellas que pertenecieran a la categoría opuesta.

Los resultados de estas pruebas fueron los siguientes.

- En primer lugar, el número de *outliers* de la mayoría de las categorías disminuyó con respecto a la utilización del *mean vector*. La diferencia es bastante baja pero se refleja una mejora. Los falsos positivos también disminuyeron pero continúan con valores similares a los de la etapa 2.
- Colocar un gran número de palabras negativas provoca que el vector obtenido no represente a la categoría de buena manera. El número de *outliers* se incrementa bastante respecto a no colocar palabras negativas. Sin embargo, si agregamos un subconjunto de palabras negativas se ve que el resultado es más confiable. Este comportamiento nos hizo pensar que el hecho de agregar muchas palabras negativas provoca que el vector generado tienda a alejarse de ellas en lugar de acercarse a las palabras positivas. Conociendo este comportamiento optamos por generar los vectores en base a las palabras positivas (pertenecientes a la categoría) solamente. Las palabras negativas las utilizaremos de forma manual, para alejar de los falsos positivos que aparezcan como resultado del algoritmo. Para incluir esta idea es necesario que el algoritmo desarrollado sea iterativo y semisupervisado, logrando de esta forma en cada iteración aprender de los errores anteriores.

Respecto a la conjugación de los verbos, los resultados que obtuvimos no fueron muy alentadores. Comenzamos con la idea de utilizar conjugación automática ya que de esa forma el algoritmo podría escalar fácilmente al incorporar nuevas palabras. Para ello utilizamos la biblioteca *pattern* [33] que según lo investigado contemplaba las necesidades que teníamos. Luego de utilizar la biblioteca vimos que las conjugaciones generadas no son las correctas, sino que se basan fuertemente en la utilización de reglas sencillas de conjugación, sin considerar casos que no las cumplen.

Considerando que el resultado no fue satisfactorio procedimos con la conjugación manual. Para ello se procedió a buscar en el diccionario de FreeLing[17, 8] las diferentes conjugaciones de cada verbo y sustituir los mismos en cada rasgo en que participaba.

Al conjugar manualmente los verbos vimos que el problema continúa presentándose. Este comportamiento tiene sentido ya que la conjugación de un verbo puede verse como una transformación dentro del espacio vectorial. Por lo tanto, conjugar los verbos es simplemente mover dentro del espacio vectorial todos los verbos, manteniendo su similitud coseno relativa.

Por lo tanto, la conjugación de los verbos no soluciona el problema de que todos los verbos están *cerca* unos de otros.

La integración con WordNet se realizó de forma sencilla y sin grandes problemas. Sin embargo, como WordNet en español no es tan completo como el de inglés, no sirve mucho para las palabras que no tenemos en el modelo vectorial. La mayoría de las palabras que no tienen vector asociado no se encuentran en WordNet o cuentan con un único synset. Esto es de esperar ya que aquellas palabras que no se encuentran en el modelo vectorial son las que menos se utilizan en el español. Considerando la falta de completitud de WordNet para el idioma español es de esperar que tampoco se encuentren en dicho recurso. Si bien en esta etapa los resultados con WordNet no fueron muy productivos, guardamos la implementación ya que podía llegar a utilizarse al tener un mayor número de palabras temporales.

#### Resultados obtenidos y conclusiones

El balance de esta etapa puede resumirse en los siguientes puntos:

- La conjugación de los verbos no es viable para resolver el problema de similitud de verbos en infinitivo. Es probable que sea necesario contar con un modelo vectorial más específico (mayor dimensión).
- El nuevo vector generado representa mejor la categoría que el mean vector utilizado anteriormente. Además permite agregar palabras negativas.
- La utilización de palabras negativas es necesario realizarla discretamente. Colocar muchas palabras provoca que el vector pierda la representación de la categoría. Surge la idea de un algoritmo semisupervisado interactivo con el usuario final.
- Realizamos la integración del algoritmo con WordNet. Los resultados no fueron útiles en esta etapa pero se cuenta con la implementación ya realizada para utilizar en instancias futuras.

### 4.4.4. Etapa 4

#### Contexto, insumos y objetivos

Esta etapa se caracterizó por estabilizar los algoritmos de clasificación automática basados en el modelo vectorial que veníamos desarrollando de las etapas anteriores.

Nos enfocamos fuertemente en definir los algoritmos de forma que sean dinámicos y parametrizables. De esta forma los algoritmos pueden evolucionar a medida que se adquieren nuevas expresiones temporales. También nos enfocamos en resolver los inconvenientes que quedaron pendientes de las etapas anteriores como ser el tratamiento de las frases y el problema de los verbos en infinitivo.

La principal diferencia de esta etapa respecto a las anteriores radica en que se agregan dos nuevos insumos que permitieron avanzar bastante en el proyecto. Dichos insumos son los siguientes:

- Corpus *Spanish Billion Words Corpus and Embeddings* (SBWCE) de Cristian Cardellino [7]. Este corpus es bastante más grande que el la Wikipedia y se utiliza para generar un nuevo modelo vectorial con mayor dimensión.
- Lista de 200 palabras temporales nuevas con aproximadamente 30 palabras clasificadas en función de todas las categorías. Para cada palabra se dispone además de

la definición que permite catalogar a la misma como temporal.

### Trabajo realizado

Los algoritmos y desarrollos de esta etapa no fueron en notebooks de Python. Se construyó una pequeña aplicación en Python que permitía ejecutar los algoritmos mediante comandos de consola. Se optó por esta alternativa para disponer de una mayor interacción con los algoritmos (por ejemplo realizar pruebas variando los parámetros).

Lo primero que realizamos en esta etapa fue construir un nuevo modelo vectorial con el nuevo corpus. Dicho modelo lo generamos con 500 dimensiones, un número mayor del que estábamos utilizando hasta el momento (100). Optamos por aumentar la dimensión del modelo con el propósito de incluir mayor información semántica en cada vector y mejorar de esta forma los resultados obtenidos. Sumado a esto, este nuevo modelo vectorial lo creamos incluyendo frases. De esta forma, aquellas expresiones que son frases pueden ser procesadas por los algoritmos que utilizan el modelo vectorial.

Para crear este modelo utilizamos el cluster de la Facultad de Ingeniería ya que el tamaño del corpus de entrada es demasiado grande para procesarlo en nuestras computadoras personales en un tiempo razonable (superaba las 24hs de cómputo). Considerando que el nuevo corpus cuenta con textos más diversificados que la Wikipedia es de esperar que el nuevo modelo vectorial pueda captar nuevos significados y usos de las palabras. Se buscó utilizar todos los parámetros disponibles para crear un modelo bien aplicado al problema planteado. Este proceso se describe con más detalles en el capitulo A del anexo.

Al evaluar el nuevo modelo vimos que presenta una calidad mejor respecto a los anteriores. Esto puede apreciarse en la tabla 4.5 donde se visualiza que en la mayoría de los conjuntos de test utilizados se obtienen resultados más precisos.

Considerando que este nuevo modelo presenta una calidad mejor respecto a los anteriores es de esperar la obtención de mejores resultados ya que las palabras están mejor representadas.

La siguiente modificación que realizamos fue la de incluir la categoría gramatical (PoSTag) en los algoritmos de clasificación. Optamos por incluir esta nueva feature a causa de los siguientes motivos:

| **Test** | **GloVe 200** | **W2V 100** | **W2V 200** | **W2V 300** | **W2V 500** |
|---|---|---|---|---|---|
| capital-common-countries | 85.1 % (291/342) | 70.8 % (242/342) | 90.5 % (277/306) | 92.8 % (284/306) | 94.1 % (288/306) |
| capital-world | 87.2 % (870/998) | 64.3 % (739/1149) | 87.9 % (878/999) | 88.3 % (981/1111) | 90.2 % (1112/1233) |
| currency | 3.5 % (3/86) | 0.0 % (0/106) | 3.8 % (2/52) | 15.4 % (8/52) | 17.3 % (9/52) |
| city-in-state | 52.0 % (632/1216) | 14.3 % (139/970) | 14.6 % (104/714) | 31.8 % (182/572) | 15.3 % (82/537) |
| family | 72.5 % (222/306) | 73.5 % (225/306) | 74.5 % (228/306) | 89.7 % (244/272) | 85.7 % (233/272) |
| gram1-adjective-to-adverb | 6.2 % (13/210) | 15.4 % (42/272) | 26.5 % (81/306) | 28.4 % (97/342) | 24.2 % (112/462) |
| gram2-opposite | 7.1 % (4/56) | 22.2 % (16/72) | 26.7 % (24/90) | 30.9 % (34/110) | 39.7 % (62/156) |
| gram5-present-participle | 26.3 % (133/506) | 66.6 % (337/506) | 70.8 % (358/506) | 78.7 % (398/506) | 74.5 % (411/552) |
| gram6-nationality-adjective | 81.6 % (1002/1228) | 72.2 % (887/1228) | 91.4 % (1123/1228) | 45.5 % (121/266) | 91.6 % (1125/1228) |
| gram7-past-tense | 9.9 % (50/506) | 18.0 % (91/506) | 19.0 % (96/506) | 22.5 % (114/506) | 27.5 % (152/552) |
| gram8-plural | 23.9 % (268/1122) | 32.1 % (318/992) | 61.0 % (644/1056) | 58.6 % (510/870) | 63.7 % (592/930) |
| gram9-plural-verbs | 22.0 % (143/650) | 37.0 % (222/600) | 44.2 % (310/702) | 45.8 % (298/650) | 47.5 % (309/650) |
| total | 50.2 % (3631/7226) | 46.2 % (3258/7049) | 60.9 % (4125/6771) | 58.8 % (3271/5563) | 64.7 % (4487/6930) |

Detrás de cada porcentaje dice cuántos resultados positivos hubo con respecto al total. El total varía de modelo a modelo porque cuando una palabra no se encuentra en el modelo entonces ese test no cuenta provocando que algunos modelos tengan más tests que otros pero sin que esto afecte el balance general

La última fila muestra el resultado global de ese modelo

GloVe 200: modelo Glove de dimensión 200

W2V 100: modelo Word2Vec de dimensión 100

W2V 200: primer modelo generado en este proyecto. Modelo Word2Vec de dimensión 200

W2V 300: Modelo Word2Vec dimension 300 descargado de SBWCE

W2V 500: Segundo modelo generado en este proyecto. Word2Vec dimension 500.

Solo los modelos W2V 200 y W2V 500 cuentan con frases.

Cuadro 4.5: Comparación Modelos Vectoriales

- La cercanía detectada en los verbos mostró que los modelos vectoriales conservan relaciones morfológicas y gramaticales además de semánticas. Incluir el PoSTag en los análisis puede aportar a que las similitudes semánticas prevalezcan sobre las gramaticales y morfológicas.
- Al analizar las categorías en las que se obtienen los mejores resultados se aprecia que sus integrantes están concentrados en pocas clases gramaticales. Aquellas categorías que tienen mayor diversidad de clases gramaticales presentan peores resultados.

Para determinar el PoSTag de cada expresión utilizamos Freeling. Optamos por Freeling debido a que produce buenos resultados en el idioma español. Para integrar Freeling con Python utilizamos el plugin generado por Mathias Laino como proyecto de grado [13].

El siguiente punto que nos enfocamos en analizar fue la forma de incluir el PoSTag en los algoritmos que teníamos. Se desarrolló una variación del *Algoritmo Vector SC - V1* que en lugar de crear un único vector por categoría crea un vector por cada categoría gramatical. Este nuevo algoritmo (*Coarse Tags clusters*) podría resumirse en el siguiente pseudocódigo:

- Por cada categoría determinar el conjunto de PoSTags a los que pertenecen sus integrantes. Esta tarea se realiza obteniendo el CoarseTag asociado a cada palabra usando FreeLing.
- Por cada PoSTag de cada categoría:
  - ⋆ Crear un vector representante en función de los integrantes con dicha categoría gramatical.
- Para cada nueva expresión a clasificar realizar los siguientes pasos:
  - ⋆ Calcular el PoSTag asociado a la expresión temporal
  - ⋆ Buscar el vector representante de la categoría asociado a dicho PoSTag. Si no existe vector asociado a dicho PoSTag la expresión no pertenece a la categoría.
  - ⋆ Calcular la SC entre el vector asociado a la expresión temporal y el vector identificado en el paso anterior. Si el vector se encuentra dentro del umbral la expresión pertenece a la categoría. En caso contrario no pertenece.

Con este cambio, la precisión del algoritmo aumentó de manera importante. El número de *outliers* se redujo a prácticamente cero (Ver tabla 4.6), logrando un gran avance respecto a la versión anterior del algoritmo.

| **Categoría** | **Cantidad de Representantes** | **Falsos Negativos (Outliers)** |
|---|---|---|
| Deíctico | 16 | 1 |
| Anclaje | 19 | 0 |
| Desplazamiento | 14 | 1 |
| Tiempo - Espacio | 9 | 0 |
| +Recurrente | 14 | 0 |
| -Recurrente | 11 | 0 |
| Orden | 17 | 0 |
| +Delimitado | 36 | 12 |
| -Delimitado | 15 | 1 |
| +Especificado | 26 | 7 |
| -Especificado | 32 | 21 |
| Puntual | 23 | 3 |
| Transformatividad | 33 | 6 |
| Duración | 19 | 2 |
| Frecuencia | 25 | 4 |

Cuadro 4.6: Outliers

Sumado a esto, el número de falsos positivos se redujo considerablemente llegando a ser cero en algunas de las categorías con mejor desempeño. Sin embargo, considerando el total de las palabras con las cuales contábamos en esta instancia el número de palabras sugeridas era demasiado bajo respecto al esperado (Ver tabla 4.7).

Respecto a las frases, en esta etapa no realizamos ningún avance significativo. De todas las frases temporales disponibles, la gran mayoría de las mismas no se encuentran en el modelo vectorial generado. Esto se debe a que las frases temporales son frecuentes en el lenguaje hablado o de narrativa. Por ejemplo, las frases *tan pronto como*, *de ahora en más*, *por los siglos de los siglos* y *esto va para largo* no son comunes de encontrar en el corpus utilizado. En caso de utilizar un corpus que contenga historias o narraciones es de esperar que dichas palabras aparezcan en el modelo. Otro factor que influye es la forma con la que Word2Vec genera las frases. Esto está explicado con mayor detalles en la sección A.3 del anexo.

Por otro lado, contar con la lista de palabras y su definición nos hizo visualizar una particularidad de la red temporal que no habíamos considerado en las etapas anteriores.

| **Categoría** | **Cantidad de Predicciones** |
|---|---|
| Deíctico | 8 |
| Anclaje | 7 |
| Desplazamiento | 4 |
| Tiempo - Espacio | 1 |
| +Recurrente | 10 |
| -Recurrente | 6 |
| Orden | 14 |
| +Delimitado | 11 |
| -Delimitado | 2 |
| +Especificado | 12 |
| -Especificado | 5 |
| Puntual | 9 |
| Transformatividad | 4 |
| Duración | 5 |
| Frecuencia | 1 |
| Tiempo - Manera | 5 |
| Individuo | 8 |
| Fase | 14 |

Cuadro 4.7: Cantidad de Palabras

En la nueva lista de palabras temporales, cada definición de palabra podía tener una clasificación diferente en los rasgos de la red temporal. Por ejemplo la palabra *abril* tiene las siguientes 2 definiciones:

- Cuarto mes del año. Tiene 30 días.
- Primera juventud.

Si analizamos el rasgo de *recurrencia*, la primer definición de *abril* sería *+Recurrente*, mientras que la segunda definición sería *-Recurrente*. Esta particularidad presenta un problema ya que el algoritmo desarrollado hasta el momento no considera múltiples significados por palabra. Cada palabra se considera de forma unitaria, englobando la totalidad de las definiciones en el vector. Por lo tanto todas las posibles definiciones de una palabras se corresponden al mismo vector y se clasifican de igual forma. Sumado a esto, en los casos en que la definición temporal de la palabra no es la más utilizada en el idioma español, se presenta el problema de que el vector representa en mayor medida el significado comúnmente usado y no tanto el significado temporal de la palabra. Esta particularidad generó que la feature de obtener sinónimos mediante WordNet tampoco pueda ser utilizada ya que habría que definir una equivalencia entre la definición de las palabras y la definición del synset provista por WordNet.

### Resultados obtenidos y conclusiones

En líneas generales, la etapa 4 fue muy productiva y permitió avanzar en forma considerable en el proyecto. Podemos resumir los hitos alcanzados en esta etapa en los siguientes puntos:

- Se logra generar un modelo vectorial más completo que permite modelar de mejor manera cada palabra. Considerando una mejor representación de la palabras es esperable que se obtengan mejores resultados. Con este nuevo modelo vectorial el problema de los verbos en infinitivo se reduce de forma considerable.
- Se avanza considerablemente en la precisión del algoritmo, logrando reducir el número de *outliers* prácticamente a cero. Lograr reducir el número de *outliers* indica un mejor modelado de la categoría.
- Se incorpora la información del PoSTag al algoritmo de clasificación de forma automática. La asignación automática permite que el algoritmo escale al incorporar

nuevas palabras.

- Se obtiene un aplicativo que mediante la ejecución de comandos por consola permite cargar palabras temporales desde un archivo en formato CSV y clasificar las mismas utilizando los algoritmos desarrollados. Contar con este aplicativo permite realizar pruebas variando los parámetros de los algoritmos. También se tiene escalabilidad al incorporar nuevas palabras temporales al proyecto.
- Se utilizó el cluster de la Facultad de Ingeniería. Si bien este hito aportó al proyecto un modelo vectorial más completo en un tiempo razonable, el aprendizaje adquirido en la utilización de esta infraestructura puede ser transmitido y utilizado en otros proyecto del grupo de PLN.

Por otro lado, quedaron algunos puntos pendientes para investigar y mejorar en las siguientes etapas del proyecto. Estos puntos pueden resumirse en:

- Encontrar una manera alternativa de procesar las frases temporales ya que el modelo actual no las incluye.
- Investigar y analizar modificaciones al algoritmo desarrollado para poder trabajar en el mismo con palabras con múltiples definiciones.
- Lograr aumentar el número de candidatos predichos para cada categoría. Si bien en esta etapa la precisión mejoró considerablemente es necesario predecir más cantidad de palabras para que el trabajo sea útil al DTLLG.

En líneas generales, esta etapa fue muy productiva ya que se logró contar con una versión básica del algoritmo. Se resolvieron muchos de los problemas presentados en la etapas anteriores y los resultados obtenidos presentaron mejores valores en lo referente a la precisión. Por otro lado, el conocimiento sobre los modelos vectoriales y la red temporal fueron incrementados, logrando una mayor comprensión de los mismos.

### 4.4.5. Etapa 5

#### Contexto, Insumos y objetivos

En esta etapa se terminó el diseño principal de los algoritmos de clasificación. Se terminaron de efectuar algunos experimentos e ideas que estaban pendientes de las etapas anteriores

refinando de esta forma los algoritmos desarrollados.

Lo primero que realizamos en esta etapa fue analizar qué insumos de los disponibles no se estaba aprovechando. El modelo vectorial y el PoSTag ya se estaban utilizando en los algoritmos y no se identificaba otra forma de sacarles provecho

La información que no se utiliza hasta el momento son las definiciones de las diferentes palabras. Una forma de utilizar dichas definiciones y que parece aportar algo al problema era utilizar con ellas la similitud de Lesk[14]. Como la similitud de Lesk relaciona palabras con semántica similar a partir de sus definiciones, es de esperar que dichas palabras tengan los mismos rasgos temporales.

Sumado a esto, la similitud de Lesk permite resolver tres aspectos que estaban pendientes.

- Las frases y aquellas palabras que no están en el modelo vectorial pueden relacionarse mediante Lesk a una expresión que si está en el modelo. Con esto pueden clasificarse las mismas utilizando el vector de aquella expresión relacionada.
- Siguiendo el mismo razonamiento, el *recall* debería mejorar ya que se pueden agregar a las categorías aquellas palabras que estén relacionadas mediante el algoritmo de Lesk a los miembros o palabras predichas de la misma.
- Por último, como Lesk trabaja sobre las definiciones se puede enfrentar el problema de las palabras con más de una definición tratándolas de forma separada.

Por lo anteriormente mencionado, la similitud de Lesk parece ser la alternativa más viable para enfrentar el problema de las palabras con varias definiciones y fue en lo que más se enfocó esta etapa. Siguiendo la misma linea, se ve que muchas definiciones incluyen sinónimos implícitamente. Por ejemplo, se define *chaval* como “Niño, muchacho”. Entonces, podemos asegurar que *chaval* es un sinónimo de tanto *niño* como *muchacho*. Dentro de las definiciones provistas por el DTLLG observamos que varios de estos patrones se repetían bastante seguido. De ahí nació la idea de parsear las definiciones en busca de ciertos patrones (reglas) que pudieran identificar sinónimos.

### Trabajo realizado

Considerando las reglas detectadas y la similitud de Lesk definimos un algoritmo que crea

un grafo dirigido de sinónimos.

Se modifican los algoritmos de clasificación para considerar esta nueva similitud. Lo que se realiza es dividir la clasificación en dos grandes fases.

La primer fase se mantiene igual a la etapa anterior y consiste en predecir nuevas palabras en función de la similitud coseno y los vectores de las palabras.

Luego se procede a "expandir" el grupo de palabras predichas agregando aquellas palabras que son similares considerando el grafo de sinónimos. Esta expansión se realiza hasta cierta profundidad en el grafo. Esta profundidad viene dada por cuántos niveles se le permite al algoritmo navegar en el grafo de similitudes. Por ejemplo, si se toma profundidad 1 solamente se usarán las expresiones que están directamente relacionadas con la expresión objetivo (están en un primer nivel de similitud respecto a esa expresión). Si se toma profundidad 2, también se consideran aquellas expresiones que son sinónimos de las expresiones del primer nivel. Esta estrategia permite llegar a una cantidad grande de palabras y expresiones. Sin embargo, al realizar la expansión se aumenta el error ya que probablemente al considerar varios niveles en el grafo se vincularán palabras que no tienen ninguna relación de similitud entre ellas.

En el algoritmo de expansión, las palabras o expresiones similares a integrantes de la categoría se agregan directamente a las palabras predichas. Por otro lado, como las palabras predichas en la primer fase no necesariamente son correctas se implementa un umbral de expansión para sus sinónimos. Este umbral define un porcentaje de palabras a las que tiene estar asociado el sinónimo para poder agregarlo como predicción. Por ejemplo, si definimos el porcentaje del umbral como 0.1 (10 %) y tenemos una palabra $x$ que es similar a una palabra *w1* predicha en la primer fase: consideraremos que $x$ es candidato a pertenecer a la categoría si $x$ está relacionado por similitud con al menos el 10 % de las palabras de la categoría (ya sea por definición o por ser predichas en la primer fase). De esta forma se reduce el arrastre de error de palabras mal clasificadas en la fase uno del algoritmo.

Este algoritmo es muy eficaz y aumenta considerablemente el número de palabras predichas respecto a la versión anterior.

Sin embargo, existen algunas palabras que no son predichas para ninguna de las categorías

de la red temporal. Para resolver este problema se agregó un algoritmo de clasificación automática que clasifica todas aquellas palabras que no fueron asignadas a ninguna categoría por las fases anteriores. Este algoritmo puede ser elegido entre *LabelPropagation*, *LabelSpreading*[29] o una red neuronal convolutiva sencilla (de tres capas). Estos algoritmos se entrenan tomando como ejemplos de entrenamiento a todas las palabras que pertenecen a cada categoría así como todas las palabras predichas. Se eligieron *LabelPropagation* y*LabelSpreading* por ser especialmente aptos para clasificación semisupervisada. Este hecho se corroboró luego al comparar el resultado de estos algoritmos con otros de clasificación automática y al observar que estos algoritmos retornaban una mayor cantidad de resultados.

Como la cantidad de ejemplos clasificados es relativamente baja no se puede entrenar estos algoritmos con una gran cantidad de valores de entrada por la conocida maldición de las dimensiones (*curse of dimensionality*). Por lo tanto redujimos la dimensión de los vectores asociados a cada palabra a valores menores a 10, usando T-SNE[38, 39]. Además tomamos como feature de entrada el PoSTag de la palabra.

Por la baja cantidad de ejemplos clasificados si la cantidad de features de entrada supera cierto límite entonces estos algoritmos no clasifican ninguna palabra. Para *LabelPropagation* y *LabelSpreading* este límite resulta ser 4 o 5 features de entrada para los datos disponibles. Para la red neuronal podemos usar hasta 7 u 8 features con resultados aceptables.

Como ya fue mencionado, estos modelos se entrenan con las palabras y expresiones ya clasificadas así como también las predicciones de las fases anteriores, lo que arrastra cierto error hacia esta fase. Debido a eso, a la baja cantidad de ejemplos clasificados y a que se pierde mucha información al reducir la dimensión de los vectores de 500 a algún valor menor a 10, no esperamos tener un gran porcentaje de aciertos en esta fase pero si por lo menos sugerir alguna categoría para cada palabra temporal.

Con esta modificación el algoritmo se compone de tres grandes etapas:

- Clasificación con similitud coseno (*Coarse Tags clusters* o *Distancia a N*)
- Clasificación con grafo de similitudes
- Algoritmo de clasificación automática para aquellas palabras que no han sido asignadas a ninguna categoría

Con esta versión del algoritmo podemos, dado un conjunto de palabras, asignarles a cada una de ellas al menos una de las categorías de la red. La certeza con que se realiza dicha clasificación depende en parte de la categoría que se está asignando y el algoritmo que lo realiza.

Por otro lado nos pareció que podía ser de utilidad para el DTLLG si además de clasificar las palabras brindadas el algoritmo pudiera sugerir nuevas palabras a incluir.

Para ello utilizamos la función *most_similar* de gensim para agregar a cada categoría aquellas palabras que son similares pero no se encuentran en la lista original de palabras temporales.

Esta funcionalidad tiene el propósito de asistir al DTLLG en la generación del corpus de palabras temporales sugiriendo nuevas palabras y expresiones.

### Resultados obtenidos y conclusiones

Con esta variación del algoritmo realizamos una serie de ejecuciones enviando los resultados recabados al DTLLG. Dichos resultados fueron evaluados por el DTLLG. La evaluación se realizó sobre las predicciones arrojadas en la primer etapa del algoritmo (similitud coseno). Estos resultados pueden apreciarse en el cuadro 4.8

Al analizar los resultados de esta etapa podemos extraer las siguientes conclusiones.

- Existen algunas categorías cuyo desempeño es muy bueno (*Anclaje, Orden, +Delimitado, Frecuencia, Individuo*). Sin embargo, algunas categorías presentan una mala calidad de los resultados (*-Delimitado, Transformatividad, Puntual*). Si analizamos estas categorías con mal desempeño vemos que las mismas están definidas por criterios muy específicos y precisos. El definir categorías de forma tan específica provoca que al momento de definir si una expresión pertenece o no a la misma se generen discrepancias entre los propios integrantes del DTLLG. Si la definición de pertenencia es difícil de realizar de forma manual es de esperar que los métodos de clasificación automáticos no obtengan resultados muy buenos.
- Notamos que algunas de las palabras son clasificadas erróneamente a causa de que existe un referente de la categoría cuya definición utilizada no es la más usual (por

| **Categoría** | **Cantidad de Palabras Categoría** | **Cantidad de Predicciones** | **Falsos Positivos** | **Precisión** |
|---|---|---|---|---|
| Deíctico | 20 | 40 | 14 | 0.65 |
| Anclaje | 33 | 65 | 10 | 0.84 |
| Desplazamiento | 19 | 26 | 16 | 0.38 |
| Tiempo-Espacio | 14 | 19 | 8 | 0.57 |
| +Recurrente | 16 | 72 | 32 | 0.55 |
| -Recurrente | 12 | 62 | 38 | 0.38 |
| Orden | 34 | 87 | 17 | 0.80 |
| +Delimitado | 37 | 113 | 7 | 0.94 |
| -Delimitado | 16 | 50 | 46 | 0.08 |
| +Especificado | 21 | 79 | 30 | 0.62 |
| -Especificado | 31 | 74 | 28 | 0.62 |
| Puntual | 26 | 99 | 60 | 0.39 |
| Transformatividad | 35 | 47 | 39 | 0.17 |
| Duración | 22 | 42 | 19 | 0.55 |
| Frecuencia | 23 | 29 | 5 | 0.83 |

Cuadro 4.8: Resultados Etapa 5

ejemplo, la palabra abril y su definición "*Años de edad de una persona joven.*" es catalogada como menos recurrente). Como el vector generado por Word2Vec engloba las utilizaciones más comunes de las palabras, en este caso no está representando correctamente la definición que se pretende utilizar. Esta situación no puede resolverse de forma directa y requiere que un humano indique que, o bien no se utilice *abril* como insumo positivo del algoritmo o que se coloquen algunos otros meses como componentes negativos. Esto requiere que la herramienta final del proyecto presente un mayor dinamismo permitiendo efectuar variaciones de los algoritmos alterando sus parámetros e insumos de entrada.

- Las similitudes encontradas en el grafo generado mediante el algoritmo de Lesk son muy buenas. Esto se debe en gran parte a que la calidad de las definiciones de las palabras aporta mucha información por ser específicas. No detectamos ningún caso en el cual el algoritmo de Lesk o las reglas propuestas relacionan palabras que no son similares.
- Los resultados arrojados por la clasificación automática con redes neuronales, LabelPropagation o LabelSpreading también son prometedores. En algunas de las expresiones clasificadas por estos algoritmos fueron predichos correctamente muchos de los rasgos que pertenecían a la misma. Considerando la escasa información de la que disponían estos algoritmos (vector de palabras y PoSTag) y los pocos ejemplos clasificados con que se contaba el resultado obtenido cubre las expectativas.
- La funcionalidad de sugerir nuevas palabras al DTLLG mediante los resultados de la función *most_similar* de *gensim* fue muy productiva. Las palabras propuestas en la mayoría de las categorías eran sugerencias que tenían sentido. Además, generó que algunas palabras que en principio fueron descartadas como temporales por el DTLLG vuelvan a considerarse. Por ejemplo, fueron sugeridos períodos religiosos y etapas arqueológicas para los cuales el DTLLG re-evaluó su pertenencia a la red temporal.

Con los resultados obtenidos hasta el momento nos parece que la investigación y experimentación se puede dar por finalizada. Se avanzó bastante en varios de los objetivos del proyecto, superando en algunas ocasiones lo esperado. Los algoritmos desarrollados se desempeñan de forma correcta para algunas categorías, siendo de esperar que al contar con más palabras en cada una de ellas se logren mejores resultados. Se cubrieron las funcionalidades esperadas por el DTLLG, agregando incluso algunas cosas que no eran

solicitadas en el alcance primario del proyecto.

# Capítulo 5

# Resultados Obtenidos

## 5.1. Análisis de resultados

Al analizar de forma global los resultados obtenidos en este proyecto, vemos que los mismos son muy buenos, superando las expectativas iniciales en algunos de los puntos. Los algoritmos desarrollados cubren en gran parte los requerimientos que necesita el DTLLG para avanzar más fluidamente en la elaboración de la red temporal.

Desde un punto de vista cuantitativo, los algoritmos desarrollados demuestran ser eficientes para las categorías *Anclaje*, *Orden*, *+Delimitado*, *Frecuencia* y *Individuo*. En estas categorías el número de falsos positivos en las expresiones que se predicen es bastante bajo. Esto permite que la asignación de estos rasgos de forma automática pueda realizarse con seguridad de que la calidad de dicha asignación será buena. Además, el número de predicciones es bastante elevado, lo que permite aumentar en pocas iteraciones el número de expresiones asociadas a dicho rasgo. En el caso de las otras categorías, la precisión obtenida con los algoritmos no es suficientemente elevada como para permitir la asignación del rasgo de forma automática. En estos casos, la revisión humana previo a la asignación del rasgo es necesaria. Estos resultados se pueden apreciar en la Tabla 5.1

Si analizamos los resultados desde un punto de vista no tan cuantitativo, los resultados obtenidos también presentan un balance positivo. Si bien los resultados numéricos muestran

| **Categoría** | **Cantidad de Palabras Categoría** | **Cantidad de Predicciones** | **Falsos Positivos** | **Precisión** |
|---|---|---|---|---|
| +Delimitado | 37 | 113 | 7 | 0.94 |
| Anclaje | 33 | 65 | 10 | 0.84 |
| Frecuencia | 23 | 29 | 5 | 0.83 |
| Orden | 34 | 87 | 17 | 0.80 |
| Deíctico | 20 | 40 | 14 | 0.65 |
| +Especificado | 21 | 79 | 30 | 0.62 |
| -Especificado | 31 | 74 | 28 | 0.62 |
| Tiempo-Espacio | 14 | 19 | 8 | 0.57 |
| Tiempo-Espacio | 14 | 19 | 8 | 0.57 |
| +Recurrente | 16 | 72 | 32 | 0.55 |
| +Recurrente | 16 | 72 | 32 | 0.55 |
| Duración | 22 | 42 | 19 | 0.55 |
| Puntual | 26 | 99 | 60 | 0.39 |
| Desplazamiento | 19 | 26 | 16 | 0.38 |
| -Recurrente | 12 | 62 | 38 | 0.38 |
| Transformatividad | 35 | 47 | 39 | 0.17 |
| -Delimitado | 16 | 50 | 46 | 0.08 |

Cuadro 5.1: Resultados Cuantitativos

un correcto desempeño para 5 categorías de las 18, la utilización que el DTLLG le dará a los algoritmos desarrollados permite obtener valor de todas las categorías. En las categorías que la precisión está en el entorno de 0.5 y 0.6 se podrán obtener resultados productivos en una utilización semisupervisada de los algoritmos. Las categorías que presentaron una precisión menor también servirán al DTLLG ya que se les brindara un universo de palabras menor en el que enfocar la clasificación manual (solo las palabras predichas), lo que agilizará el trabajo.

Si analizamos los algoritmos desarrollados podemos ver que los resultados que arrojan los mismos dependen del contexto en que se encuentren. Podríamos resumir las asociaciones de contexto-algoritmo de la siguiente forma:

- ***Coarse Tags clusters***: Algoritmo útil para realizar la expansión de las categorías cuando se dispone de pocos elementos clasificados. Esto se puede lograr ya que gran parte de la información que utiliza el algoritmo proviene del modelo vectorial. Este algoritmo es útil cuando se dispone de palabras que son utilizadas comúnmente, y por lo tanto el vector asociado a la misma la modela de buena forma. También es necesario que la definición asociada a la palabra con la que se está tratando sea la utilización más usual de la misma, de forma que el vector obtenido del modelo se pueda asociar a la definición.

- ***Distancia a N***: Considerando que este algoritmo se basa también fuertemente en el modelo vectorial, los contextos de utilización son iguales que los del algoritmo Coarse Tags clusters.

- ***Expansión por sinónimos***: Este algoritmo no utiliza el modelo vectorial, sino que se basa fuertemente en la definición de las expresiones. Este algoritmo es útil cuando las expresiones temporales con las que se están trabajando contienen definiciones completas. Usar las definiciones de las expresiones permite distinguir entre distintos significados de la misma y asignar rasgos distintos a cada uno de ellos.

- ***Red neuronal, LabelPropagation y LabelSpreading***: Estos algoritmos son útiles cuando el volumen de datos ya clasificados es grande. Por lo tanto, se deberían utilizar en las últimas iteraciones de la clasificación

Si analizamos el trabajo realizado, podemos apreciar que la metodología realizada fue

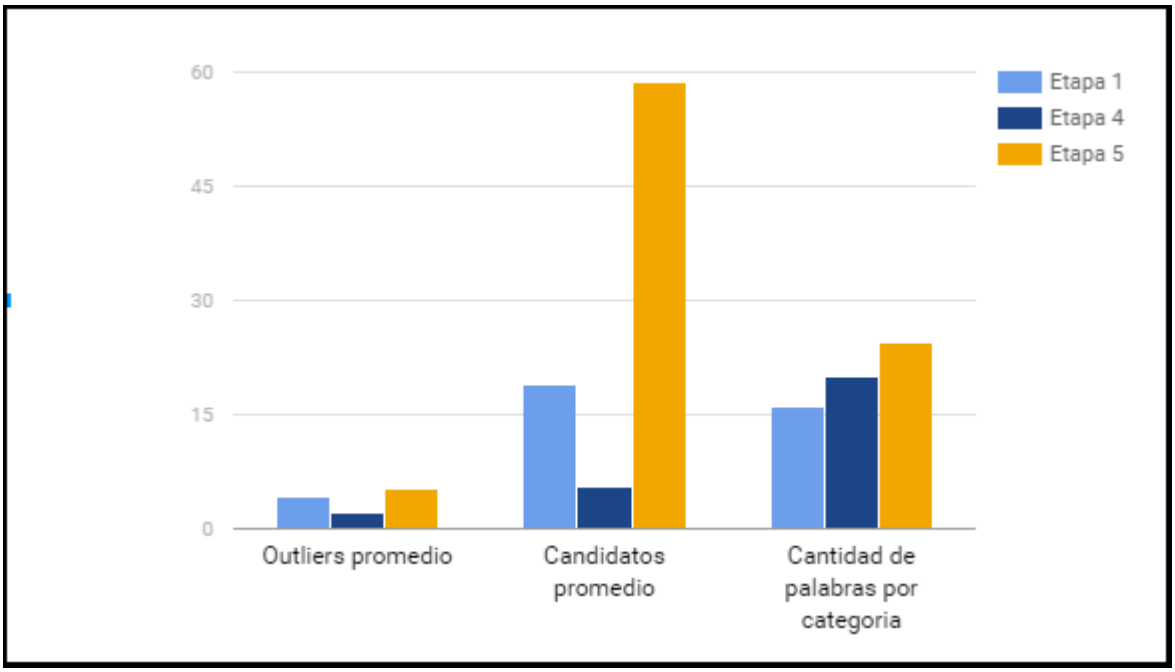


Figura 5-1: Evolución de Resultados por Etapas

buena ya que se obtuvieron resultados positivos de una problemática difícil de enfrentar. Por otro lado, los cambios de etapas presentan una evolución que se ve reflejada en la Figura 5-1.

En la gráfica 5-1 se observa que en la etapa 4 se puso énfasis en reducir la cantidad de falsos negativos u outliers en las categorías buscando la forma de representar las categorías de la mejor manera. Los outliers volvieron a aumentar en la etapa 5 principalmente porque había más palabras por categoría, sin embargo la mayoría de estos outliers quedaron muy cerca del umbral. También se observa como la etapa 5 estuvo más centrada en mejorar el recall del algoritmo, produciendo más candidatos para cada categoría.

En la gráfica 5-2 se nota que durante una buena parte del proyecto se estuvo trabajando con un universo de palabras temporales bastante pequeño que se incrementó luego de la etapa 3.

Considerando la variedad de algoritmos y sus diferentes contextos podemos ver que el trabajo realizados es bastante completo ya que permite trabajar en diferentes situaciones. Esto da libertad de trabajo y flexibilidad al DTLLG para las tareas de clasificación.

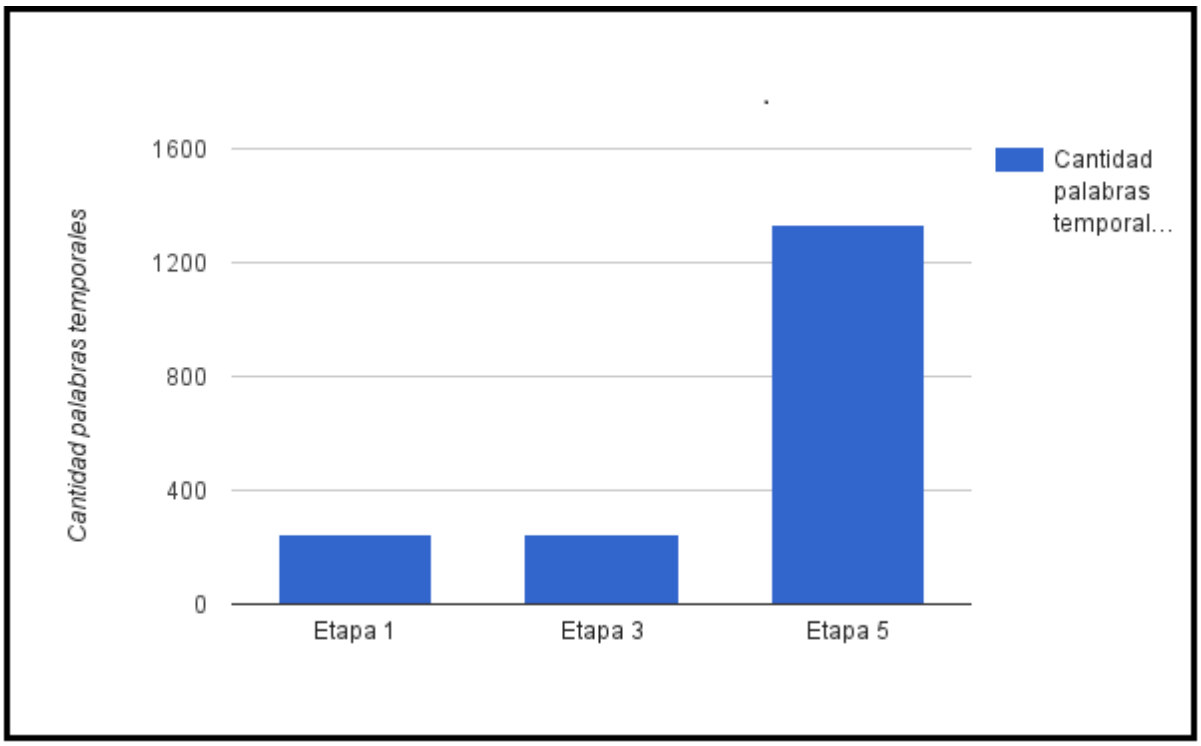


Figura 5-2: Cantidad de palabras Temporales

En líneas generales, considerando los buenos valores obtenidos en las categorías *Anclaje*, *Orden*, *+Delimitado*, *Frecuencia* e *Individuo* sumado a la variedad de algoritmos desarrollados podemos concluir que los resultados obtenidos son muy buenos y que cubrieron las expectativas iniciales del proyecto.

# Capítulo 6

# Herramienta Web para la clasificación de palabras temporales

Como mencionamos anteriormente, en la etapa 5 damos por culminada la investigación e experimentación del proyecto ya que los algoritmos que desarrollamos presentan las cualidades esperadas. Sin embargo, para que los algoritmos desarrollados sean de más provecho y más fáciles de usar para el DTLLG decidimos desarrollar una interfaz web que da acceso a las funciones implementadas. De esta forma se facilita la definición del problema, la clasificación y el análisis de los resultados de la misma.

Otra razón que influye en esta decisión es el hecho que el conjunto de expresiones temporales aún está en construcción; por lo que seguramente en el futuro se agregaran más expresiones para ser clasificadas. Por lo tanto, para que el proyecto sea de utilidad para los integrantes del DTLLG; los mismos deberán poder clasificar las nuevas expresiones que se agreguen a la red temporal.

Por otro lado, en las pruebas realizadas se vio que el algoritmo funciona mejor si se utiliza de forma semisupervisada, en un enfoque similar al propuesto por Active Learning. Es decir, se generan diferentes instancias de clasificación y entre instancia e instancia se determina manualmente si los resultados son correctos. De esta forma, mediante varias iteraciones se van asignando rasgos temporales a las palabras en función de los datos iniciales más los resultados de iteraciones anteriores.

Por los puntos anteriores decidimos implementar una herramienta web que sirva para realizar estas acciones de manera más cómoda e intuitiva. En este capítulo detallaremos los principales componentes de la herramienta y sus funcionalidades.

Cabe destacar que la herramienta implementada no está directamente ligada a la temporalidad; sino que se corresponde a una herramienta de clasificación semántica. Ese aspecto hace que la herramienta pueda ser utilizada por el DTLLG o el grupo PLN para otros problemas de clasificación semántica.

## 6.1. Diseño de la solución

En esta sección se describe el producto generado a lo largo del proyecto. Se muestran detalles del mismo sin entrar demasiado profundo en su implementación. Se muestra una idea básica del modelo de clases, las funcionalidades principales de la interfaz web así como una descripción de los algoritmos de clasificación implementados.

### 6.1.1. Backend - Modelo de clases

En esta sección describiremos las clases más importantes implementadas en nuestro modelo.

Por un lado tenemos las clases *Category* y *Word* que representan categorías y expresiones temporales respectivamente. Estas clases se pueden vincular principalmente por dos relaciones: una expresión puede pertenecer a una categoría o no. Cuando una expresión pertenece a una categoría estará asociada a la misma como positiva, y si no pertenece a la categoría estará asociada como negativa. Cabe destacar que hasta que una palabra no haya sido clasificada como positiva o negativa para una categoría, no estará asociada a esa categoría baja ninguna de las dos relaciones. Como es evidente, las palabras provistas por el DTLLG en la definición de los siguientes rasgos son agregadas en la carga inicial como positivas.

La clase *Word* contiene, además de la palabra o frase a la cual representa, información adicional de la expresión como lo son su definición, su PosTag y su CoarseTag.

Por otro lado están las clases relacionadas con la clasificación de las expresiones temporales. Dentro de estas se destacan *ClassificationInstance*, *ClassificationResult* y *ClassificationResultMember*. La primera representa a una instancia de clasificación particular, o sea una fase o etapa de clasificación. Esta instancia tendrá asociada un *ClassificationResult* por cada categoría definida, el cual guarda todas las expresiones predichas para dicha categoría. Estas predicciones se representan con la clase *ClassificationResultMember*.

Para ver más detalles acerca del modelo de clases se puede consultar la sección B.1 del anexo.

El backend está desarrollado en Python-Django con una base de datos PostgreSQL. Elegimos Python por ser el lenguaje más comúnmente utilizado en proyectos de Machine Learning.

### 6.1.2. Interfaz web

En esta sección describiremos la interfaz web desde el punto de vista del usuario final de la misma (integrante del DTLLG). Mostraremos las funcionalidades que presenta y cómo se utilizan sin entrar en los detalles de la implementación.

Las principales funcionalidades de la interfaz web son las siguientes:

- *Cargar palabras y categorías*: La herramienta permite al usuario cargar categorías junto con las palabras que pertenecen a esas categorías así como también palabras que no pertenecen a ninguna categoría. Esto puede realizarse mediante archivos en formato CSV que contienen la palabra y su definición. Si no se especifica la definición entonces no podrá usarse el algoritmo que trata de detectar sinónimos.
- *Modificar las categorías de la red temporal*: En varias de las reuniones que tuvimos con integrantes del DTLLG se nos comunicó que las categorías de la red pueden llegar a cambiar a medida que van haciendo pruebas y cuenten con más expresiones temporales con sus rasgos asignados. La interfaz web permite eliminar y agregar categorías. También permite editar qué palabras son consideradas como positivas y negativas para cada categoría.
- *Clasificar las palabras en función de las categorías definidas*: La herramienta permite

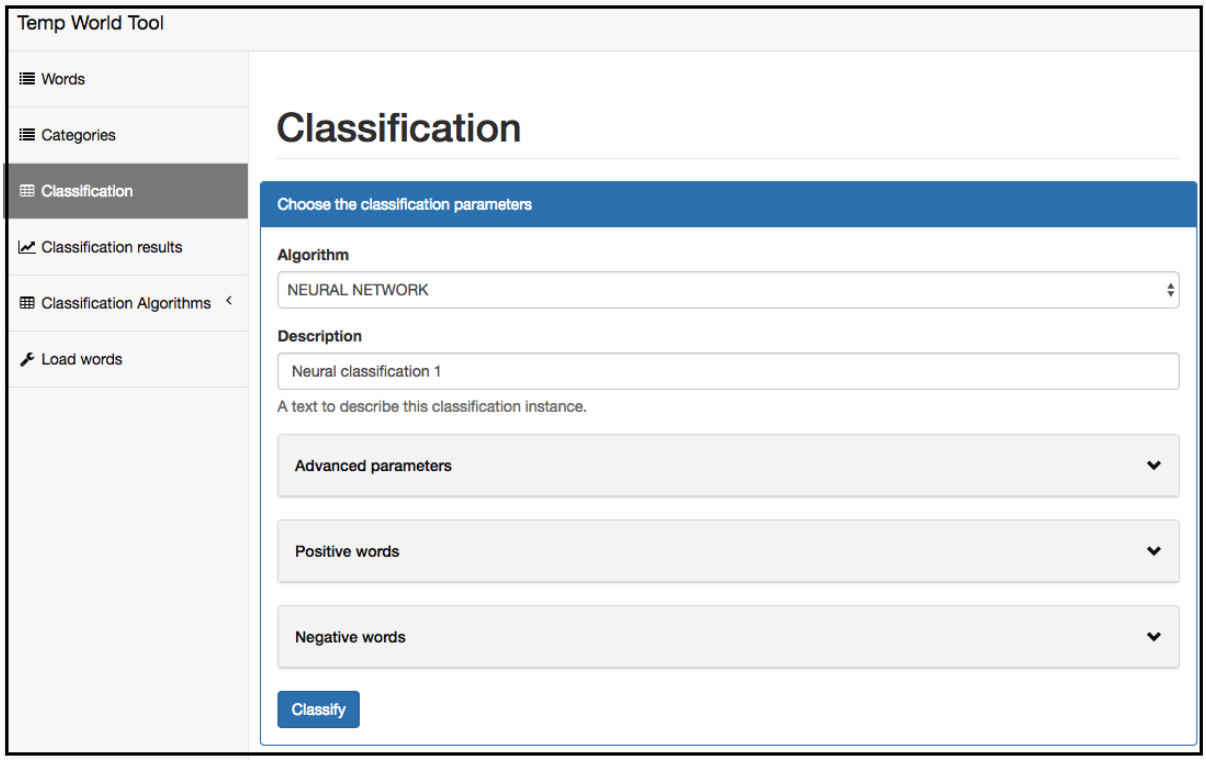


Figura 6-1: Ejecución de Algoritmos

ejecutar a demanda los algoritmos de clasificación definidos, que se detallan en la sección 6.2. A diferencia de las etapas anteriores, en las cuales los algoritmos se concatenan ejecutándose uno detrás del otro; en esta herramienta los mismos se pueden efectuar de forma independiente. Tomamos esta decisión ya que queremos que el error que se genera en fases anteriores no se arrastre a fases siguientes. Además se permite setear el valor de los parámetros y definir con qué conjunto de palabras se trabajará. Esto significa que se puede no considerar algunas palabras como positivas si se cree que no van a aportar positivamente al resultado. Un ejemplo de esto es cuando se utiliza la palabra *abril* con el significado de *juventud*. Este significado de abril pertenece a la categoría *No recurrente*. Sin embargo, el vector asociado a *abril* no refleja esa acepción ya que no es muy frecuentemente usada. En este caso puede ser útil no considerar *abril* como positivo al clasificar nuevas palabras para el rasgo *No recurrente*. Una imagen de como se visualiza en la herramienta web los diferentes algoritmos y sus parámetros pueden apreciarse en la Figura 6-1.

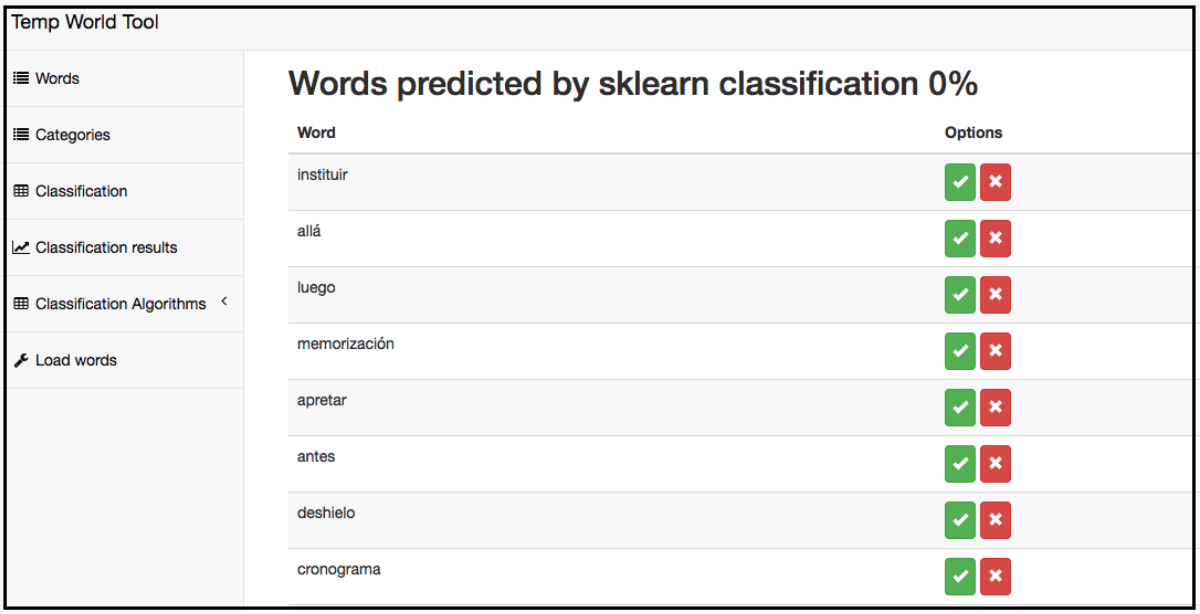


Figura 6-2: Evaluación de Resultados

- *Evaluar los resultados*: Cada instancia de clasificación crea un conjunto de resultados (palabras o frases asociadas a una categoría) que deben evaluarse manualmente. Esto puede hacerse viendo todas las palabras clasificadas para una categoría dada y es la forma recomendada para agregar palabras a cada categoría. El usuario tiene la posibilidad de marcar cada resultado como correcto, erróneo o dejarlo sin marcar. Todos los resultados no marcados como correctos ni erróneos no alteran la composición de las categorías, o sea no se agregan a la categoría (ni al conjunto de palabras positivas ni negativas). Pero las palabras o expresiones marcadas pasan a formar parte de los conjuntos respectivos (positivos o negativos) de cada categoría. Ejemplo de cómo se visualiza la evaluación de los resultados puede apreciarse en la Figura 6-2.
- *Visualizar asignación de rasgos para palabras*: La herramienta permite la visualización de qué rasgos tiene una palabra o qué palabras pertenecen a un rasgo (Ver Figura 6-3).

La interfaz web contiene una sección *Words* en la cual se muestra información acerca de las palabras temporales registradas y también permite algunas acciones sobre las mismas.

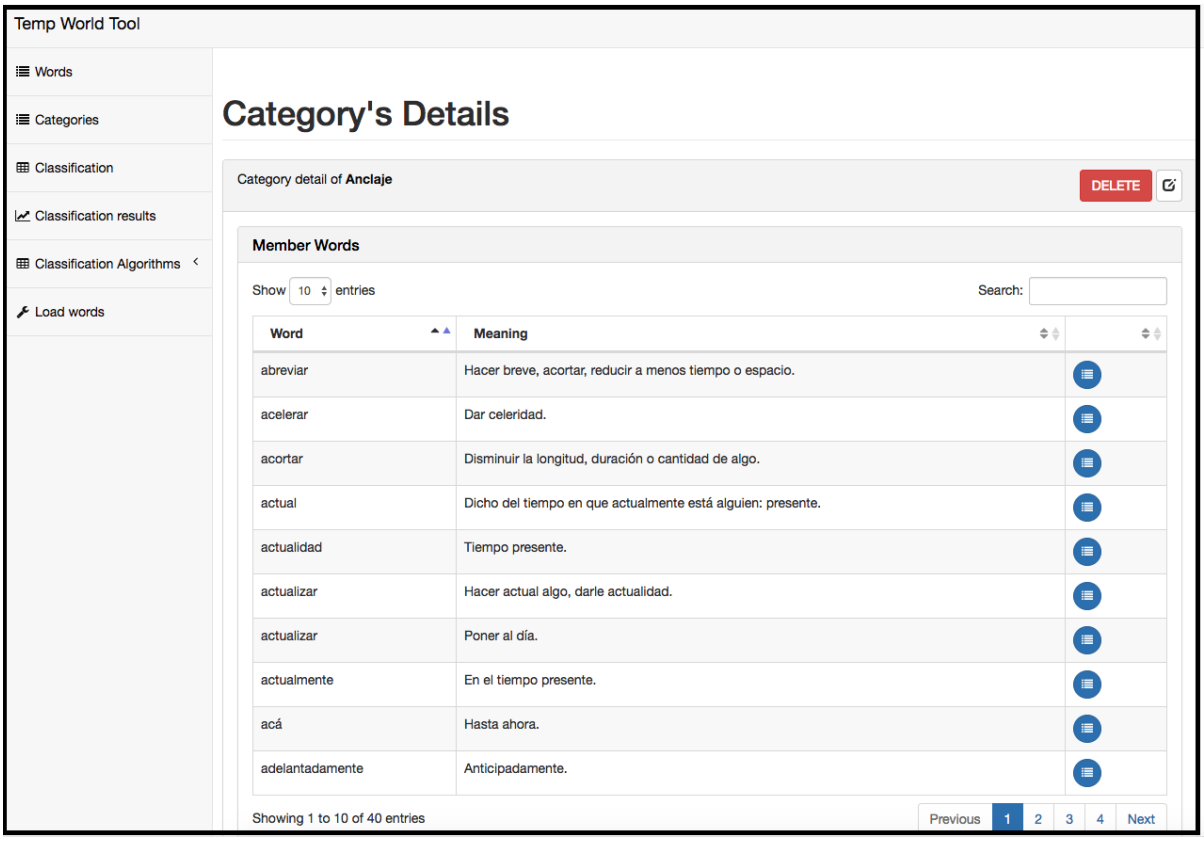


Figura 6-3: Detalles de Categoría

En la sección *Categories* se muestran las categorías del proyecto y se permite agregar, eliminar y/o modificar las mismas.

La sección *Classification* permite iniciar una nueva etapa de clasificación con alguno de los algoritmos desarrollados. Los resultados de estas etapas de clasificación pueden verse en la sección *Classification results*.

Por último en la sección *Management* se permite realizar otras acciones de gestión como los son la carga de nuevas palabras o categorías.

## 6.2. Algoritmos de clasificación

En esta sección se describen los algoritmos disponibles con una descripción general de su funcionamiento y una breve explicación de en que casos son útiles o recomendados. Además se detallan los parámetros que admiten en caso que un usuario entendido prefiera modificar los valores por defecto de los mismos.

### 6.2.1. Coarse Tags clusters

Este es el algoritmo principal, el que se fue desarrollando y testeando a lo largo de todo el proyecto. Para cada categoría propone nuevas palabras que están cerca en el espacio vectorial. Para esto toma un vector representativo de la categoría y mide la distancia de ese vector a cada palabra objetivo. Las palabras que están más cerca que cierto umbral son propuestas como candidatos a pertenecer a la categoría.

Aparte de las palabras positivas también se pueden usar palabras negativas de una categoría con el fin de alejar los resultados de esas palabras negativas.

Este algoritmo separa su problema en un problema distinto para cada CoarseTag, de ahí su nombre. Esto significa que se genera un vector representativo de la categoría para cada CoarseTag distinto y se compara cada palabra objetivo con el representante correspondiente a su CoarseTag. Esto hace que si algún CoarseTag no está presente en

los ejemplos clasificados entonces no se van a presentar resultados de palabras con ese CoarseTag.

Este algoritmo es especialmente útil en las primeras etapas para generar un conjunto de ejemplos clasificados más grande ya que no necesita demasiados ejemplos clasificados.

Solo hay un parámetro que puede usarse para este algoritmo y es el umbral que se usa para considerar a una palabra como candidata o no. Este umbral tiene un valor de 0.45 por defecto y es más restrictivo cuanto más alto sea. La sugerencia es utilizar el valor 0.45 en las primeras etapas de clasificación de forma de generar muchos candidatos para agregar a las categorías. A medida que se van agregando más palabras a las categorías se puede ir aumentando el valor del umbral de forma de obtener menos resultados pero más precisos.

### 6.2.2. Expansión por sinónimos

Este es el único algoritmo que no usa el modelo vectorial, sino que se basa exclusivamente en las definiciones de las expresiones y el grafo de similitud (o sinónimos) creado a partir de ellas. El algoritmo propone expresiones que son "similares" a expresiones que pertenecen a la categoría. Esta similitud se define en base a cuántos pasos tiene el camino más corto de una expresión que pertenece a la categoría y la expresión objetivo, en el grafo.

El algoritmo cuenta con dos parámetros: el primero es la profundidad hasta la cual se le permite navegar en el grafo de similitudes. Si este valor es -1 entonces se considerará el grafo completo. Si el camino entre dos palabras es mayor que este valor entonces no se toma como válido.

El segundo parámetro permite especificar el porcentaje de palabras que deben ser "similares" a una palabra para considerarla candidata. O sea, si una categoría tiene 20 palabras positivas y el valor de este parámetro es 0.1 (10 %) entonces para que una palabra sea candidata deberá ser similar a al menos 2 palabras. Esto pretende evitar incluir palabras que solamente son similares a una sola palabra. Se puede jugar con el parámetro definiendo distintos valores para obtener candidatos más o menos seguros.

Este algoritmo es muy útil en todas las etapas de un proyecto, pues no requiere muchos

ejemplos iniciales y va a sugerir palabras nuevas aun cuando haya muchas palabras en la categoría. Especialmente se recomienda ejecutarlo entre ejecuciones de otros algoritmos. Cabe destacar que este algoritmo generará mejores resultados si las definiciones provistas de las palabras son más específicas y detalladas.

### 6.2.3. Distancia a N

Este algoritmo es una alternativa al *Coarse Tags cluster* pero que puede ser útil en etapas tempranas así como tardías de un proyecto. El algoritmo propone palabras que están a una distancia dada de al menos $N$ palabras positivas de una categoría. Esto permite obtener buenos resultados para categorías que no tienen un núcleo bien definido en el espacio vectorial, sino que tienen una disposición más bien dispersa en varios grupos.

Al considerar menos palabras que el total de positivas de una categoría, se pueden usar distancias pequeñas (valores altos de similitud) para obtener resultados más exactos.

Los parámetros del algoritmo son el valor de $N$ y la distancia mínima a la cual deben estar las palabras candidatas.

### 6.2.4. Clasificación con Red Neuronal

Definimos una red neuronal que puede ser usada para clasificar aquellas palabras que no pertenecen a ninguna categoría. El algoritmo toma todas las palabras sin ningún rasgo asignado y trata de asociarle al menos un rasgo. Este algoritmo no es muy útil inicialmente dado que hay muy pocas palabras clasificadas, pero se espera que tenga buenos resultados cuando el conjunto de expresiones clasificadas sea más grande.

La red neuronal toma como features de entrada el vector que representa a una palabra por lo que depende del modelo vectorial y no puede clasificar aquellas palabras que no se encuentren en el mismo.

La red usa una capa de convolución y dos capas densas o *Fully Connected* para producir su resultado.

Este algoritmo no tiene ningún parámetro.

### 6.2.5. Clasificación con LabelPropagation y LabelSpreading

También definimos dos algoritmos de clasificación que son *LabelSpreading* y *LabelPropagation* implementados en la librería SKLearn [29] especialmente para casos de clasificación semisupervisada. Ambos algoritmos toman como entrada una versión reducida del vector que representa a la palabra además de, opcionalmente, el CoarseTag de la palabra. Por la baja cantidad de ejemplos, solo obtuvimos resultados con hasta 5 features lo que limita mucho el potencial de estos algoritmos. Para reducir la dimensión de los vectores usamos PCA. Al reducir tanto las dimensiones de los vectores se pierde mucha información. De hecho analizando las matrices generadas por PCA se puede ver que reducir la dimensión a 5 pierde el 86 % de la información contenida en los vectores por lo que el resultado esperado no es demasiado optimista.

Los parámetros para estos algoritmos son la cantidad de dimensiones a las que se reducirán los vectores y si se quiere usar el CoarseTag o no.

### 6.2.6. Sugerencias de palabras usando "most similar" de Gensim

Por último, este algoritmo sugiere nuevas palabras que no están aún catalogadas como temporales. Para ello utiliza la función *most_similar* que provee Gensim sugiriendo las palabras del modelo vectorial que están más cerca de las palabras positivas de cada categoría.

El parámetro que se puede usar para este algoritmo es cuántas palabras se desea que se sugieran. El algoritmo puede ser interesante para ayudarle al DTLLG a agregar palabras a su conjunto de expresiones temporales.

## 6.3. Conclusiones de la herramienta web

Al analizar las cualidades y funcionalidades incluidas en la herramienta web podemos concluir que la misma será de gran utilidad para el DTLLG.

Esta herramienta permitirá a los integrantes del DTLLG expandir de forma dinámica la asignación de rasgos de la red temporal. El hecho de contar con dinamismo es un aspecto fundamental de la herramienta considerando que muchos de los insumos y conceptos abarcados en la red temporal se encuentran en período de construcción, por lo que pueden variar.

El hecho de que la herramienta contiene algoritmos que cubren diferentes contextos permite abarcar categorías heterogéneas, como es el caso de la red temporal. Este punto da gran flexibilidad al DTLLG para la asignación de rasgos a las nuevas palabras ya que permite que se adapten a las particularidades de la categoría o la cantidad de palabras ya clasificadas.

Por otro lado, la funcionalidad de sugerir nuevas palabras temporales también será de utilidad para los integrantes del DTLLG ya que permite que evalúen tópicos que en una primera instancia tal vez no fueran considerados como temporales.

Esta herramienta fue instalada y presentada a los integrantes del DTLLG. La primer impresión que tuvieron fue muy buena, diciendo que la veían muy útil ya que les facilitaría la tarea de asignación de rasgos en la red temporal.

Por último, la utilización de la herramienta no está destinada exclusivamente a la temporalidad, lo que permite utilizar la misma en otros proyectos del DTLLG o del grupo de PLN de la Facultad de Ingeniería.

Considerando los puntos anteriores creemos que la herramienta desarrollada es buena ya que permite englobar los experimentos realizados durante el transcurso del proyecto y ayudará a la creación de la red temporal.

## 6.4. Bibliotecas y herramientas

En esta sección detallaremos algunas de las bibliotecas y recursos informáticos que se utilizaron para implementar la herramienta web o alguno de los experimentos. Las herramientas serán presentadas de forma genérica, enfatizando que cualidades de las mismas fueron explotadas en el proyecto.

### 6.4.1. Reducción de dimensiones con T-SNE y PCA

Para algunas tareas se debe reducir la dimensión de los vectores del modelo para poder usarlos. Esto sucede, por ejemplo, para visualizar las palabras ya que solo podemos graficar 2 o 3 dimensiones pero no 100 o 500. Además puede ser usado cuando los vectores se usan como features en una etapa de clasificación y no se desea tener demasiadas features por no contar con demasiados ejemplos.

Un algoritmo muy usado para reducir dimensiones es PCA (Principal component analysis) [31] que aplica SVD a los vectores y se queda con las dimensiones de mayor varianza, o sea las que son más determinantes.

Otra técnica, propuesta más recientemente es t-SNE [39] que es mejor que PCA cuando se usa para visualizar vectores, o sea, cuando se reduce a 2 o 3 dimensiones [38].

Las dos técnicas, PCA y t-SNE están implementadas en SKLearn [32].

T-SNE trata de mantener las relaciones de cercanía de los puntos en la dimensión alta en la dimensión baja. Su función de costo no es convexa por lo que cada invocación puede dar resultados diferentes. Se recomienda usar otro algoritmo de reducción como PCA [31] antes de aplicar T-SNE para reducir el ruido y acelerar el tiempo de cómputo. En este proyecto se usó PCA para este propósito.

Se usó este algoritmo para visualizar los vectores generados por Word2Vec o GloVe para tener una idea de la distribución de las categorías de palabras en el espacio vectorial. Se usó PCA para reducir la dimensión de los vectores para poder usarlos como feature en algunos de los algoritmos de clasificación implementados.

### 6.4.2. FreeLing

FreeLing [22, 17, 24, 8, 2, 23] es una librería open source escrita en C++ que permite hacer análisis sintáctico de textos en varios idiomas, entre ellos el español. Entre otras cosas FreeLing sirve para obtener Part of Speech Tags (PoSTag) o tokenizar enunciados que fueron las funcionalidades que se usaron en este proyecto. En este proyecto se usó la versión 3.1 de FreeLing.

Para poder usar FreeLing desde Python se uso el adaptador de Matias Laino publicado en GitHub [13] que soporta la versión 3.1 de FreeLing.

### 6.4.3. WordNet

WordNet [36] es una base de datos léxica en inglés que agrupa palabras en conjuntos de sinónimos llamados *synsets* de los cuales cada uno expresa un concepto diferente. WordNet proporciona definiciones cortas ejemplos de uso para estas palabras. Además almacena las relaciones léxicas y semánticas que hay entre los *synsets*. WordNet puede ser visto como una combinación de diccionario y tesauro.

Por estas funcionalidades es usado mucho en tareas de procesamiento de lenguaje o lingüística computacional. En especial, es usado con frecuencia para desambiguar el sentido de las palabras (word sense disambiguation o WSD) cuando se quiere asignar un concepto a una palabra en cierto contexto.

WordNet ha sido publicado libremente bajo una licencia BSD y puede ser descargado o consultado en línea.

### 6.4.4. Gensim

Gensim [27] es una librería en Python diseñada para obtener información semántica a partir de documentos de texto. Para ello implementa varios modelos para representar texto plano de una forma que se pueda realizar análisis sobre ellos y define una interfaz para realizar operaciones sobre los mismos. Entre los modelos implementados está Word2Vec

(descrito en sección 3.1.4) que fue el que usamos nosotros. Gensim no soporta GloVe, pero si el modelo vectorial creado por alguna implementación de GloVe se guarda en formato de texto plano y se le realizan modificaciones menores entonces puede ser cargado por Gensim como si fuese Word2Vec.

Aparte de poder cargar modelos guardados en formatos de texto, Gensim permite guardar y cargar modelos en formato binario, ocupando menos espacio y siendo más rápido el proceso de carga del modelo.

Se detalla a continuación las prestaciones que Gensim brinda para trabajar con modelos Word2Vec.

Gensim viene de “generate similar” y su función principal es encontrar palabras o frases similares a ciertas palabras o frases dadas. Para esto provee la función *"most_similar"* que dada una lista de palabras o frases que se llaman ‘positivas’ devuelve las palabras o frases más similares a estas. Además permite pasar palabras o frases catalogadas como ‘negativas’, las que pueden modificar la forma en cómo el modelo encuentra las palabras más similares. En este caso el resultado serán palabras similares, o sea cercanas en el plano vectorial a las palabras ‘positivas’ pero poco similares o lejanas a las palabras o frases negativas.

La lejanía o cercanía entre dos palabras se calcula tomando la distancia coseno entre los vectores asociados a dichas palabras. Esta distancia es configurable. Entre otras distancias que analizamos se destaca la distancia euclidiana cuya diferencia con la distancia coseno es que mide la distancia entre dos puntos en el espacio y por lo tanto su resultado depende de la norma de los vectores mientras que la distancia coseno mide el ángulo de separación entre dos vectores y por lo tanto no depende de la norma de los vectores. En el caso de Word2Vec la norma de un vector depende de cuantas veces aparece la palabra en el corpus.

En este proyecto trabajamos con vectores normalizados por lo que ambas distancias dan el mismo resultado y por eso se optó por mantener la distancia coseno en la mayoría de las pruebas.

Gensim provee una función para normalizar los vectores y al mismo tiempo ahorrar espacio de memoria que puede ser importante ya que modelos con muchas palabras y de

dimensiones altas como en nuestro caso suelen ocupar mucho espacio en memoria (unos 2GB de RAM después de normalizar los vectores en nuestro caso)

Otra funcionalidad de Gensim que usamos para generar nuestro modelo vectorial es su modelo de frases. Gensim permite encontrar frases en un corpus de enunciados y luego utilizar estas frases para crear un modelo vectorial que contenga palabras y frases. En el capitulo A del anexo se detalla cómo se encuentran las frases contando cuantas veces aparece cada palabra y cuantas veces aparecen juntas.

Gensim también tiene algunos corpus en inglés como el Brown que se pueden usar directamente, pero en este proyecto no se usó ninguno porque no había ninguno en español.

# Capítulo 7

# Conclusiones y trabajo a futuro

## 7.1. Conclusiones

Al finalizar el proyecto y evaluar el trabajo desarrollado podemos concluir que el balance del mismo fue muy positivo. Los resultados obtenidos cubren las expectativas iniciales del proyecto, logrando como resultado final una herramienta web que será de gran utilidad para la asignación de rasgos a los integrantes del DTLLG. Además, en algunas categorías se lograron obtener buenos resultados mediante clasificación automática. Considerando los escasos ejemplos ya clasificados con los que contábamos, obtener estos resultados fue un hito muy importante.

El proyecto se desarrolló sin grandes inconvenientes, logrando avances significativos entre etapa y etapa de experimentos. Estos avances se lograron en gran parte debido a que muchas de las hipótesis en las que se fundaban los experimentos fueron correctas. En proyectos de este estilo, donde la investigación y experimentación tienen un rol tan grande se corre el riesgo de pasar gran parte del mismo abordando líneas que terminan fuera de los resultados esperados. En nuestro caso, la gran mayoría de las hipótesis de trabajo desarrolladas aportaron información y valor a los algoritmos de clasificación, hecho que permitió alcanzar buenos resultados en algunas de las categorías.

Un factor fundamental de este proyecto fue la utilización del modelo vectorial de pala-

bras. Este insumo permitió agregar información semántica a las palabras y expresiones temporales de la red; información muy importante para los algoritmos de clasificación desarrollados.

Por otro lado, ajeno al problema de la red temporal, el hecho de trabajar con dichos modelos vectoriales fue un aspecto que vimos positivo y productivo de este proyecto. Los modelos vectoriales son un insumo que últimamente ha ganado notoriedad. El trabajar con un recurso de actualidad agregó valor al proyecto ya que pudimos familiarizarnos con su utilización y generación, conocimientos que pueden ser utilizados en futuros proyectos.

Otro factor que nos parece positivo destacar del proyecto fue la utilización de la similitud de Lesk para un problema que no es el usual. El algoritmo de similitud de Lesk usualmente se utiliza para la desambiguación de palabras. En este caso se utilizó para realizar clasificación semántica.

El hecho de que el proyecto sea en cierta manera interdisciplinario agrega valor al mismo. En el área del PLN es importante que los lingüistas sientan interés por los aportes que la informática pueda generar de forma de proponer nuevos problemas y proyectos que permitan avanzar en el área. En el caso de este proyecto, creemos que la ganancia fue para ambas partes.

La herramienta web desarrollada cubre bastante bien las necesidades del DTLLG en el marco de la red temporal. Permitirá clasificar, evaluar y visualizar expresiones temporales en función de los rasgos. Sumado a esto, se les brindará la posibilidad de incluir nuevas expresiones temporales e inclusive modificar los rasgos; funcionalidades que le dan a la herramienta escalabilidad y flexibilidad. Obtener esta herramienta fue un hito muy importante del proyecto ya que facilitará al DTLLG la elaboración de la red temporal.

Además de la herramienta web, el proyecto dejó a los integrantes del DTLLG información sobre la red temporal que deberán analizar. El modelo vectorial sugirió candidatos a expresiones temporales que no habían sido considerados en una primera instancia por el DTLLG (por ejemplo algunos períodos religiosos). Estos candidatos a expresiones temporales serán evaluados por los integrantes del DTLLG para ver si corresponde o no incluirlos. Por otro lado, los resultados de los experimentos permitieron detectar categorías que presentaban mejores resultados que otras. Esta diferencia de performance entre categorías se debe en gran medida al grado de especificación que se tiene en su

definición. El hecho de que la clasificación automática presente tan bajos resultados podría indicar que la definición de la categoría es demasiado específica y que debería revisarse. Este tipo de información genérica de la red temporal es muy útil para el DTLLG ya que les permite profundizar en características de la misma.

Respecto al relacionamiento interdisciplinario, el mismo fue muy bueno durante todo el transcurso del proyecto. La disposición de los integrantes del DTLLG para reunirse y evaluar los resultados parciales obtenidos fueron claves para poder avanzar en el proyecto.

En líneas generales, creemos que el proyecto fue bueno y terminó aportando conocimientos tanto al grupo de PLN como al DTLLG. El hecho de trabajar en un proyecto no trivial fue un gran atractivo para nosotros ya que nos enfrentó a desafíos interesantes de resolver. Este punto junto a la utilización de herramientas y conceptos de vanguardia fueron algunos de los factores de motivación más importantes. Se logró adquirir conocimiento importante sobre la utilización de los modelos vectoriales y su aplicación a un problema específico de clasificación semántica. Se lograron buenos resultados de clasificación automática en las categorías *Anclaje*, *Orden*, *+Delimitado*, *Frecuencia* e *Individuo*. Por último, se logró un producto web que será de ayuda al DTLLG en el contexto de la red temporal. Por todos estos puntos creemos que el balance del proyecto fue bueno.

## 7.2. Trabajo a futuro

Al finalizar este proyecto surgen algunos puntos que estaría bueno de profundizar pero que no estaban incluidos en el alcance del mismo o no se realizaron por falta de tiempo. Estos puntos pueden tratarse en futuros proyectos y pueden resumirse en lo siguiente:

- *Evaluar funcionalmente los algoritmos de redes neuronales.* Una vez que se disponga de un volumen de datos ya clasificado pueden utilizarse los algoritmos de redes neuronales para la clasificación de nuevos elementos. En este proyecto no pudimos evaluar los algoritmos debido a que el volumen de datos nunca fue suficiente para el entrenamiento de las redes.
- *Evaluar el producto final con otros problemas de clasificación semántica.* Los algoritmos desarrollados nunca utilizan el concepto de temporalidad, sino que se basan

en los vectores, PosTags y definiciones de las expresiones. Esta particularidad de los algoritmos hacen que parezca viable su utilización en otros contextos. Sería interesante evaluar si esta suposición es verdadera.

- *Sacar ventaja de las relaciones que puede haber entre distintas categorías.* Se podría investigar la posibilidad de incorporar información de categorías opuestas entre sí o categorías que son subconjuntos de otras para obtener resultados más exactos.

# Bibliografía


[1] Shilpa Arora and Sachin Agarwal. Active learning for natural language processing. Literature review, Carnegie Mellon University, 2008.

[2] Jordi Atserias, Bernardino Casas, Elisabet Comelles, Meritxell González, Lluís Padró, and Muntsa Padró. Freeling 1.3: Syntactic and semantic services in an open-source nlp library. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC 2006)*, Genoa, Italy, May 2006. ELRA.

[3] Satanjeev Banerjee. Adapting the lesk algorithm for word sense disambiguation to wordnet. Master's thesis, University of Minnesota, December 2002.

[4] Edward Loper Bird, Steven and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[5] Eugenio Martínez Cámara, Miguel A. García Cumbreras, M. Teresa Martīn Valdivia, and L. Alfonso Ureña López. SINAI-EMMA: Vectores de palabras para el análisis de opiniones en twitter. Technical report, Universidad de Jaén, Septiembre 2015.

[6] Cesar Cardellino. Questions for Word2Vec testing. https://cs.famaf.unc.edu.ar/~ccardellino/SBWCE/questions-words_sp.txt. Accedido: 26-01-2017.

[7] Cristian Cardellino. Spanish Billion Words Corpus and Embeddings, March 2016.

[8] Xavier Carreras, Isaac Chao, Lluís Padró, and Muntsa Padró. Freeling: An open-source suite of language analyzers. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, 2004.

[9] Universitat de Barcelona. Corpus ancora. http://clic.ub.edu/corpus/en. Accedido: 31-01-2017.

[10] Gensim. Word2Vec. https://radimrehurek.com/gensim/models/word2vec.html. Accedido: 12-12-2016.

[11] Google. Word2vec training. https://code.google.com/archive/p/word2vec/. Accedido: 01-09-2016.

[12] INCO. DLPLN Seminario - Aprendizaje profundo aplicado al Procesamiento de Lenguaje Natural. https://eva.fing.edu.uy/course/view.php?id=879. Accedido: 13-02-2017.

[13] Matias Laino. Librería de extensiones para NLTK. https://github.com/matiaslaino/inco-pln-nltk-extensions. Accedido: 27-09-2016.

[14] 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*, SIGDOC '86, pages 24–26, New York, NY, USA, 1986. ACM.

[15] Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In Roser Morante and Wen tau Yih, editors, *CoNLL*, pages 171–180. ACL, 2014.

[16] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, March 2015.

[17] Marina Lloberes, Irene Castellón, and Lluís Padró. Spanish FreeLing Dependency Grammar. In *Proceedings of 7th Language Resources and Evaluation Conference (LREC'10)*, La Valletta, Malta, May 2010.

[18] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 142–150, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 01 2013.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 10 2013.

[21] J.P. Ordoñez. Introducción a las redes neuronales artificiales. https://jpordonez.wordpress.com/2008/08/06/introduccion-a-las-redes-neuronales-artificiales/. Accedido: 08-02-2017.

[22] Lluís Padró. Analizadores Multilingües en FreeLing. *Linguamatica*, 3(2):13–20, December 2011.

[23] Lluís Padró, Miquel Collado, Samuel Reese, Marina Lloberes, and Irene Castellón. Freeling 2.1: Five years of open-source language processing tools. In *Proceedings of 7th Language Resources and Evaluation Conference (LREC'10)*, La Valletta, Malta, May 2010.

[24] Lluís Padró and Evgeny Stanilovsky. Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*, Istanbul, Turkey, May 2012. ELRA.

[25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[26] Radim Řehůřek. Making sense of word2vec. https://rare-technologies.com/making-sense-of-word2vec/, 12 2014. Accedido: 27-09-2016.

[27] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[28] sklearn. k-Means. http://scikit-learn.org/stable/modules/clustering.html#k-means. Accedido: 27-09-2016.

[29] sklearn. Label Propagation. http://scikit-learn.org/stable/modules/label_propagation.html. Accedido: 27-09-2016.

[30] sklearn. Multiclass classification. http://scikit-learn.org/stable/modules/multiclass.html. Accedido: 27-09-2016.

[31] sklearn. PCA - principal component analysis. http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. Accedido: 27-09-2016.

[32] sklearn. t-SNE. http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. Accedido: 27-09-2016.

[33] T. De Smedt and W. Daelemans. Pattern for python. *Journal of Machine Learning Research*, 13:2031–2035, 2012.

[34] TensorFlow. Vector representation of words. https://www.tensorflow.org/tutorials/word2vec/. Accedido: 08-02-2017.

[35] Jörg Tiedemann. Parallel Data, Tools and Interfaces in OPUS. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[36] Princeton University. WordNet. http://wordnet.princeton.edu, 2010. Accedido: 19-11-2016.

[37] L. van der Maaten. t-SNE. https://lvdmaaten.github.io/tsne/. Accedido: 27-09-2016.

[38] L.J.P. van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15, October 2014.

[39] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9, Nov 2008.

# Apéndice A

# Generación del modelo vectorial

## A.1. Introducción

Inicialmente se trabajó con un modelo vectorial brindado por el grupo de PLN cuyos vectores tenían dimensión 100. Esto es relativamente poco para realizar profundas investigaciones según comentarios de otros investigadores ya que generalmente se suelen utilizar dimensiones mayores a 200[16].

Además de esta particularidad, el modelo inicial no contaba con frases; lo que generaba que las frases temporales que pretendíamos clasificar no pertenecían al modelo.

A causa de estos motivos fue necesario crear nuevos modelos vectoriales.

## A.2. Preprocesamiento

Se empezó por descargar el ultimo *dump* de la Wikipedia en español conteniendo las páginas y artículos de la misma página web. Wikipedia cuenta con distintas selecciones con distintos tipos de texto entre los cuales también se destacan los metadatos, versiones anteriores de artículos presentes y foros de discusión. Nosotros elegimos solo páginas y artículos actuales ya que son textos formales de los que se espera un ruido bastante menor.

Esta elección la comparten la gran mayoría de ejemplos e investigaciones consultadas.

Luego se dividieron por enunciados los textos extraídos de este conjunto de páginas y artículos. A su vez se eliminaron los símbolos que no pertenecen al lenguaje español y se transformó todo el texto a minúsculas.

Se decidió eliminar los símbolos que no pertenecen al lenguaje español (como acentos que aparecen en nombres de otros idiomas) ya que las palabras que las contienen no son de interés para el problema planteado al no pertenecer al idioma español.

Aparte se reemplazaron los números por la palabra 'num' al concluir que no aporta aspectos positivos el hecho de diferenciar entre los distintos números. De este modo tanto el número '9' como el '1992' se reemplazaron por la palabra 'num'

Por último se eliminaron también los signos de puntuación dejando a cada enunciado como una simple lista de palabras.

## A.3. Creación de frases

Luego del pre-procesamiento se prosiguió a crear frases. Para esto se utilizo la librería Gensim[27] con la cual se puede entrenar un modelo que encuentra frases al aplicarlo a una lista de enunciados. Estos modelos son capaces de encontrar bigramas contando cuántas veces aparecen dos palabras juntas y separadas; considerando a estas palabras como frase si cumplen la fórmula propuesta por Mikolov et. al. para frases[20] (Ver fórmula A.1).

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)} \tag{A.1}$$

En esta ecuación, el puntaje para cada frase equivale a la cantidad de apariciones de ambas palabras juntas menos un coeficiente $\delta$ ( usado para prevenir demasiadas frases) dividido entre la cantidad de apariciones de cada palabra multiplicados entre sí. Si este puntaje es mayor que cierto umbral entonces estas palabras se considerarán frase. Dicho umbral es un parámetro configurable usando la librería Gensim.

Otro mecanismo para limitar el número de frases es poner un mínimo de apariciones que

deben tener esas palabras juntas para ser considerados una frase. El valor por defecto para este parámetro es 5, pero nosotros elegimos 10 como la cantidad mínima de apariciones ya que sino aparecían muchas frases con poco sentido.

Si este tipo de entrenamiento se aplica varias veces usando el resultado de la ejecución anterior se pueden formar n-gramas. Se decidió aplicar esto dos veces ya que cada iteración consume más recursos de memoria y cómputo que la anterior y además introduce cierto porcentaje de ruido que va creciendo con cada iteración. De esta forma fue posible encontrar los bigramas, trigramas e incluso algunos de los cuatrigramas más comunes.

El resultado esperado era encontrar la gran mayoría de las frases presentadas como temporales con este método pero el resultado fue distinto. Frases como 'al cabo' y 'ni bien' no fueron encontradas en el modelo y se concluyó que eso se dio debido a que son frases usadas mayormente en historias y diálogos, y no en artículos como los de la Wikipedia. Otras frases como 'por los siglos de los siglos' quedaron afuera del modelo por ser demasiado largas.

## A.4. Entrenamiento del primer modelo vectorial

Para entrenar el modelo las decisiones se basaron fuertemente en resultados de otros experimentos anteriores como el de Radim Rehurek[26] así como el que realizaron Levy, Goldberg y Dagan[16].

Lo primero que se intentó fue aumentar la dimensión de los vectores a 400. Esto resultó imposible en una máquina personal(8Gb de RAM) ya que el entrenamiento del modelo consume mucha memoria RAM y el sistema operativo terminaba matando al proceso del entrenamiento por consumir demasiada memoria.

Por lo tanto se crearon vectores con dimensión 200.

El resto de las decisiones se tomaron respetando los artículos antes mencionados usando el modelo skip-gram como el mejor (dicho por Mikolov mismo al presentar el modelo Word2Vec[19]) con *negative sampling*, que es más eficiente que *hierarchical softmax* para palabras frecuentes [20, 16]).

Este modelo mejoró considerablemente los resultados de modelos anteriores, al menos en las pruebas automatizadas realizadas. Sin embargo contaba con demasiadas frases no tanto comunes que seria mejor eliminar del mismo.

## A.5. Segundo modelo, usando cluster FING y SBWCE

Debido a los límites de hardware de nuestras máquinas personales se decidió usar el cluster de la FING para tratar de mejorar el modelo un poco más. Estas mejoras se trataron de obtener mediante tres ajustes. Primero, agrandar el corpus, diversificando los textos. Esto conlleva que ciertos usos de palabras que no son frecuentes en Wikipedia sean considerados en el modelo. Para esto usamos el corpus publicado en la pagina SBWCE[7] cuyo tamaño supera en más del doble al de la Wikipedia. Este corpus contiene mayúsculas y minúsculas por lo que el único cambio que se le realizó fue el de transformar todas las letras a minúsculas. El corpus, aparte de incluir un *dump* de la Wikipedia, también contiene partes de los corpus Ancora [9] y OPUS Project[35], que incluye libros, noticias y documentos de las Naciones Unidas entre otras.

El segundo ajuste fue el de reducir las frases. Esto se logró por un lado duplicando el umbral usado para que dos palabras se consideren como una frase. Además; se duplicó la cantidad mínima de apariciones de estas frases. Esto a su vez se compensa por agrandar el corpus, con lo que seguramente haya más apariciones de cada frase.

El tercer ajuste fue el de aumentar la dimensión del modelo generado a 500. Este valor, aunque empírico, fue decidido considerando una mezcla de factores como la memoria que se necesita para cargar el modelo en tiempo de ejecución (apróx. 2Gb de RAM) y la precisión que se gana aumentando la dimensión.

## A.6. Resultados comparativos

Se realizaron tests de precisión con todos los modelos para tener una forma de evaluarlos.

Estos tests se basan en la implementación de la librería Gensim que prueba la precisión del modelo en base a analogías como por ejemplo “Montevideo es a Uruguay como París es a . . . ?”.

Para esto se usó un archivo descargado de la misma página web del SBWCE. Dichos resultados pueden apreciarse en el cuadro A.1.

Se observa que los resultados mejoraron bastante con los modelos generados en este proyecto con respecto a los modelos iniciales. También parece claro que aumentar la dimensión del modelo vectorial mejora la precisión. Por otro lado notamos que los modelos iniciales parecen tener un vocabulario mas amplio ya que se pudieron ejecutar mas tests (comparando los valores entre paréntesis).

Podemos concluir que el segundo modelo generado en este proyecto es el mas apto para nuestro problema ya que presenta la mejor precisión de todos y aparte presenta un vocabulario bastante grande.

| **Test** | **GloVe 200** | **W2V 100** | **W2V 200** | **W2V 300** | **W2V 500** |
|---|---|---|---|---|---|
| capital-common-countries | 85.1 % (291/342) | 70.8 % (242/342) | 90.5 % (277/306) | 92.8 % (284/306) | 94.1 % (288/306) |
| capital-world | 87.2 % (870/998) | 64.3 % (739/1149) | 87.9 % (878/999) | 88.3 % (981/1111) | 90.2 % (1112/1233) |
| currency | 3.5 % (3/86) | 0.0 % (0/106) | 3.8 % (2/52) | 15.4 % (8/52) | 17.3 % (9/52) |
| city-in-state | 52.0 % (632/1216) | 14.3 % (139/970) | 14.6 % (104/714) | 31.8 % (182/572) | 15.3 % (82/537) |
| family | 72.5 % (222/306) | 73.5 % (225/306) | 74.5 % (228/306) | 89.7 % (244/272) | 85.7 % (233/272) |
| gram1-adjective-to-adverb | 6.2 % (13/210) | 15.4 % (42/272) | 26.5 % (81/306) | 28.4 % (97/342) | 24.2 % (112/462) |
| gram2-opposite | 7.1 % (4/56) | 22.2 % (16/72) | 26.7 % (24/90) | 30.9 % (34/110) | 39.7 % (62/156) |
| gram5-present-participle | 26.3 % (133/506) | 66.6 % (337/506) | 70.8 % (358/506) | 78.7 % (398/506) | 74.5 % (411/552) |
| gram6-nationality-adjective | 81.6 % (1002/1228) | 72.2 % (887/1228) | 91.4 % (1123/1228) | 45.5 % (121/266) | 91.6 % (1125/1228) |
| gram7-past-tense | 9.9 % (50/506) | 18.0 % (91/506) | 19.0 % (96/506) | 22.5 % (114/506) | 27.5 % (152/552) |
| gram8-plural | 23.9 % (268/1122) | 32.1 % (318/992) | 61.0 % (644/1056) | 58.6 % (510/870) | 63.7 % (592/930) |
| gram9-plural-verbs | 22.0 % (143/650) | 37.0 % (222/600) | 44.2 % (310/702) | 45.8 % (298/650) | 47.5 % (309/650) |
| total | 50.2 % (3631/7226) | 46.2 % (3258/7049) | 60.9 % (4125/6771) | 58.8 % (3271/5563) | 64.7 % (4487/6930) |

Detrás de cada porcentaje dice cuántos resultados positivos hubo con respecto al total. El total varía de modelo a modelo porque cuando una palabra no se encuentra en el modelo entonces ese test no cuenta provocando que algunos modelos tengan más tests que otros pero sin que esto afecte el balance general.

La última fila muestra el resultado global de ese modelo.

GloVe 200: Modelo Glove de dimensión 200.

W2V 100: Modelo Word2Vec de dimensión 100.

W2V 200: Primer modelo generado en este proyecto. Modelo Word2Vec de dimensión 200.

W2V 300: Modelo Word2Vec dimension 300 descargado de SBWCE.

W2V 500: Segundo modelo generado en este proyecto. Word2Vec dimension 500.

Solo los modelos W2V 200 y W2V 500 cuentan con frases.

Cuadro A.1: Comparación de Modelos Vectoriales

# Apéndice B

# Modelo de la aplicación web

En esta sección describiremos el modelo diseñado para la aplicación web.

Se describe cada clase implementada así como aspectos generales (por ejemplo el manejo del modelo vectorial y particularidades de la implementación de la clasificación).

## B.1. Clases del modelo

Podemos separar las clases del modelo en tres grupos: en la primera situamos la clase *Word,* que representa a las expresiones temporales y su información asociada, junto con sus clases vinculadas *FreelingCoarseTag* y *FreelingPosTag.* En el segundo grupo situamos la clase *Category* que representa una categoría o rasgo temporal. En el último grupo situamos las clases que tienen que ver con la clasificación de las expresiones temporales.

A continuación describiremos las clases mencionadas con sus particularidades y decisiones de diseño.

- *Word*
  Esta clase representa a las expresiones temporales y una instancia de la misma se identifica por la palabra o frase representada más su definición. O sea solo existe una instancia para cada par "palabra, definición". Esto significa que una palabra

puede tener varias instancias de *Word* distintas si es que tiene varias definiciones distintas asociadas. La necesidad de tratar a las expresiones temporales de esta forma viene dado que el DTLLG asigna rasgos distintos (a veces opuestos) para diferentes definiciones de la misma palabra. De esta forma se permite, por ejemplo, clasificar "abril" como *No recurrente*, haciendo referencia a su significado de "juventud", y al mismo tiempo clasificar "abril" como *Recurrente* haciendo referencia al mes de abril. Aparte de la definición de la expresión, también se guarda el lema y un conjunto de Postags y Coarsetags obtenidos con FreeLing[17, 8]. Estos tags pueden ser usados como parámetro adicional al vector del modelo vectorial a la hora de clasificar expresiones temporales.

- *FreelingCoarseTag* y *FreelingPosTag*
  Estas clases representan a los Postags y Coarsetags que pueden tener las palabras y solo guardan información del tag representado.
- *Category*
  La clase *Category* representa las categorías del proyecto, es decir, los rasgos temporales a asignar a las palabras. Contiene listas de expresiones (*Word*) positivas, negativas y neutras. Las positivas son las palabras que pertenecen a la categoría; las negativas son las que no pertenecen mientras las neutras son para las que no corresponde indicar si pertenecen o no. Al usuario de la aplicación solamente se le permite agregar palabras al conjunto negativo y positivo mientras que el conjunto de palabras neutras existe para ser compatible con la definición del DTLLG.
  La interfaz web permite agregar, eliminar o modificar las categorías definidas inicialmente.
- *VectorModel*
  Representa a un modelo vectorial de cierto tipo (Word2Vec o GloVe) y con cierta dimensión (cantidad de valores para cada entrada). Guarda, aparte de tipo y dimensión, la dirección del archivo del modelo vectorial a cargar.
- *ClassificationInstance*
  Esta clase representa a una etapa de clasificación y por lo tanto es el punto de partida para tanto el proceso de clasificación como para el análisis de resultados de la misma. Guarda una referencia al modelo (*VectorModel*) usado así como el algoritmo empleado y la descripción proveída por el usuario. También guarda la fecha en la cual fue creada.

Por otro lado, contiene una instancia de *ClassificationResult* por cada categoría del proyecto.

- *ClassificationResult*
  Representa al resultado para una etapa de clasificación asociada a una categoría y por lo tanto contiene la lista de expresiones predichas para dicha categoría en la etapa de clasificación (*ClassificationInstance*) asociada. Estas expresiones predichas pueden ser de dos tipos distintos dependiendo del algoritmo con el que fueron creados. Si el algoritmo se basa en la definición de la expresión temporal entonces se crea una instancia de la clase *ClassificationResultMember*, que referencia a una instancia de *Word*. Por otro lado, si el algoritmo usa únicamente el modelo vectorial y no la definición de las palabras, entonces se crea una instancia de *ClassificationResultOtherWords* ya que en este caso el resultado no se puede asociar a un par "palabra, definición" sino más bien a una palabra con cualquiera (o todos) de los significados asociados.
- *ClassificationResultMember*
  Esta clase representa a una palabra predicha por algún algoritmo para cierta categoría. De esta forma tiene asociado una instancia de *Word* y pertenece a un *ClassificationResult*. En esta clase también se guarda el resultado de la evaluación del resultado, o sea si el usuario lo identificó como correcto o no.
- *ClassificationResultOtherWords*
  Esta clase es igual a la anterior pero en vez de apuntar a instancias de *Word*, tiene un atributo de tipo *String* que guarda la expresión predicha.

En la figura B-1 se muestra un esquema del modelo implementado

## B.2. Manejo del modelo vectorial

Como la carga del modelo vectorial desde un archivo puede demorar un tiempo considerable, es deseado hacer esta acción una única vez, al menos si el modelo es usado con frecuencia. Por esta razón implementamos una clase que sigue el patrón *Singleton* que guarda una referencia al modelo cargado con Gensim y que se usa en todas las interacciones con el modelo. De esta forma el modelo se carga una única vez y se mantiene en memoria

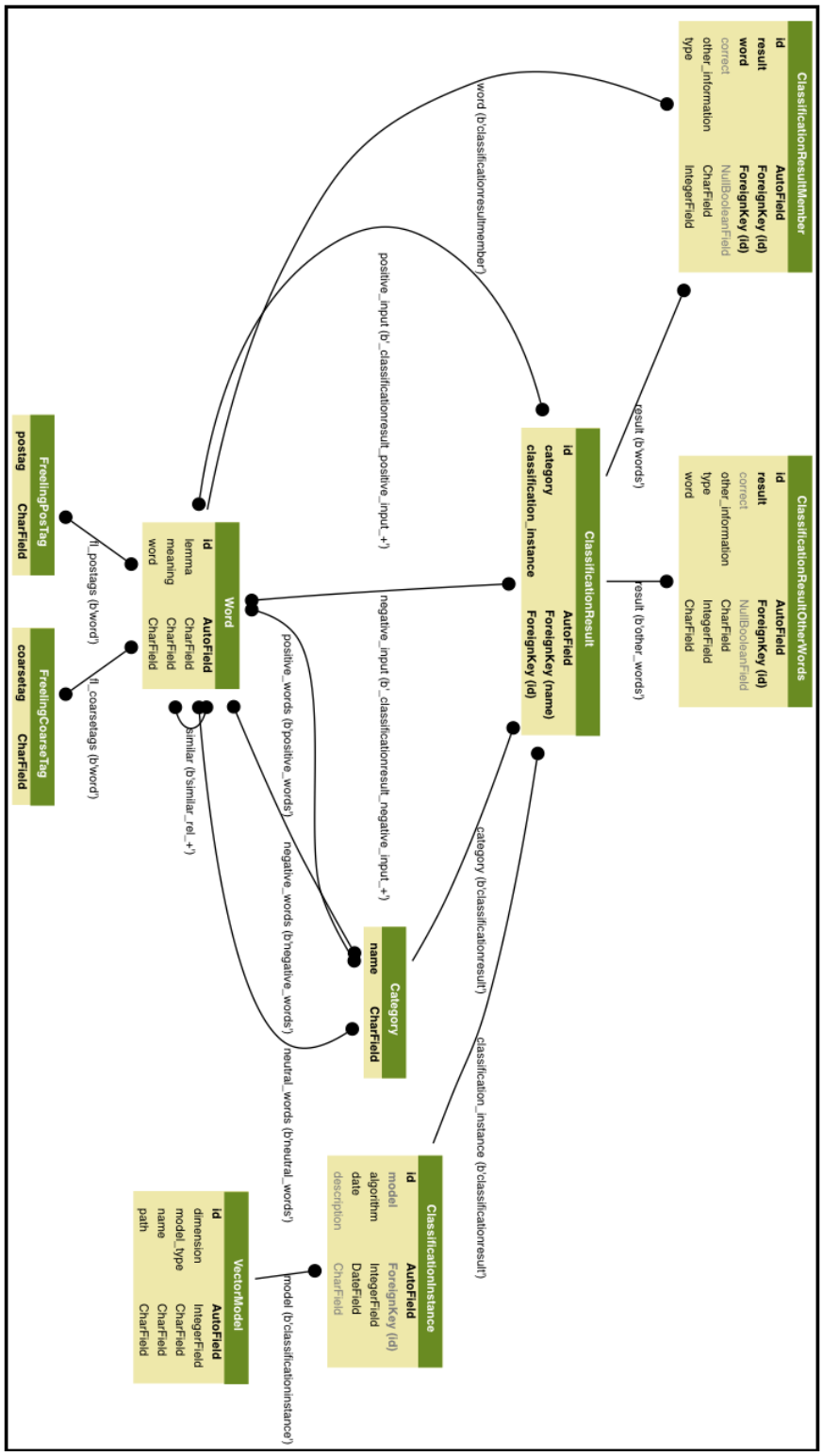


Figura B-1: Diagrama de entidades

después.

La clase *VectorModel* guarda información del modelo vectorial en la base de datos. Si el usuario de la aplicación desea cambiar el modelo vectorial usado, por ejemplo usar GloVe en vez de Word2Vec entonces se sustituye el modelo vectorial anterior con el nuevo, o sea en cada momento solo se guarda un modelo en memoria.

## B.3. Manejo de etapas de clasificación

Como una etapa de clasificación puede demorar varios minutos y puede usar un gran porcentaje de una CPU común decidimos que solo se puede ejecutar un proceso de clasificación a la vez. Esto no es una limitante para el proyecto actual ya que la tarea de los integrantes del DTLLG se centrará más en evaluar los resultados de estas etapas de clasificación por el enfoque iterativo del proyecto. Es decir, su tarea se centrará en decidir, para cada expresión predicha para alguna categoría, si fue correctamente predicha o no.

Esta ejecución se ejecuta en un hilo en *background* para de esta manera no bloquear el servidor web mientras se está ejecutando una etapa de clasificación.

## B.4. Interfaz única para acceder desde consola o interfaz web

El proyecto cuenta con una serie de funciones en un archivo (*shell.py*) que pueden ser accedidos con las distintas funcionalidades de la interfaz web. La idea de tener todas las funciones centralizadas es que se puede acceder tanto desde la web como desde la línea de comandos, ejecutando exactamente el mismo código.

De esta forma se permite testear, en caso de ser necesario, todas las funciones desde la línea de comandos.