Benemérita Universidad Autónoma de Puebla

Facultad en Ciencias de la Computación

Tesis Maestría

Similitud Semántica Textual para Asistentes Virtuales

Tesista:
Magali Fong Juárez

Director:
Dr. David Eduardo Pinto Avendaño

Codirector:
Dr. Franco Rojas López

Agradecimientos

Índice general

Índice de figuras

Índice de algoritmos

Resumen

Un asistente virtual es un software capaz de reconocer el lenguaje natural utilizado por el usuario, esto permite establecer una conversación para respuestas preguntas, hacer recomendaciones o realizar acciones solicitadas, aprovechando la capacidad de almacenamiento y procesamiento que disponen los ordenadores y múltiples dispositivos electrónicos. Con la finalidad de mejorar la experiencia de uso dotan al asistente virtual de una voz y aspecto humano.

Actualmente numerosos sitios web y establecimientos de todo el mundo cuentan con asistentes virtuales que ayudan a sus usarios y clientes a responder dudar o a guiarles en el proceso de selección y compra de un producto o servicio.

En este proyecto de tesis se evaluó las medidas de similitud semántica textual Word2Vec, GloVe y FastText, de ahí obtener una medida con mayor precision para aplicarlo en un chatbot de preguntas frecuentes.

Capítulo 1

Introducción

En este capítulo, se hablará sobre los Asistentes Virtuales (AV), así como las preguntas de investigación, objetivos y la estructura del documento.

1.1. Asistentes Virtuales

Los AV se han vuelto en gran medida imprescindibles para que las empresas ganen reconocimiento en el mercado competitivo de hoy, pueden conectarse con sus clientes e interactuar con ellos de manera personal. Con el potencial de los AV para proporcionar un servicio al cliente como nunca antes, las empresas pueden aumentar las ventas. Como resultado, los AV pueden brindar oportunidades para mejorar a las empresas a alcanzar el crecimiento comercial y obtener ganancias financieras. No sólo las empresas sino también los clientes se beneficia con esta tecnología. Se eliminan las molestias de esperar durante largas horas para ponerse en contacto con los ejecutivos de atención al cliente. Además, pueden proporcionar respuestas a los clientes incluso durante horas no operativas. Debido a las respuestas rápidas y la disponibilidad 24x7.

Cuando los AV iniciaban, no podían dirigir las conversaciones. Hoy, han evolucionado para convertirse en modelos sofisticados. Sin embargo, a veces aún carecen de la comprensión de la intención y el lenguaje del usuario. Por lo tanto, deben estar bien entrenados para comprender el contexto del usuario en el lenguaje humano.

El PLN es lo que permite a los AV comprender sus mensajes y responder de manera adecuada. Cuando envía un mensaje, por ejemplo “hola”, es el PLN el que le informa al AV que ha publicado un saludo, lo que a su vez le permite aprovechar sus capacidades de Inteligencia Artificial (IA) para obtener una respuesta adecuada. En este caso, el AV probablemente responderá con un saludo de regreso. El PLN ayuda a proporcionar contexto y significado a las entradas de usuario basadas en texto para que la IA pueda encontrar la mejor respuesta.

1.2. Preguntas de investigación

- ¿Es posible identificar las medidas de SST para acoplarlas y tener un buen rendimiento en el apoyo de Asistentes Virtuales?
- ¿Es posible mejorar el mecanismo de Asistentes Virtuales mediante la utilización de SST, mejorará los resultados con respecto a similitud léxica textual?

1.3. Objetivos

El objetivo general de esta tesis es implementar un algoritmo de Similitud Semántica Textual aplicado a un Asistente Virtual.
A continuación se en listan los objetivos específicos:

- Determinar el dominio de aplicación e identificar un conjunto de datos asociados a dicho dominio llevar a cabo la investigación.
- Evaluar medidas de Similitud Semántica Textual (SST) presentes en la literatura.
- Proponer un algoritmo SST basada en medidas de similitud, adaptada al dominio de aplicación.
- Evaluar el rendimiento de la propuesta.

1.4. Estructura del documento

Este documento de tesis se compone por 5 capítulos, lo cuales se encuentran estructurados de la siguiente manera:

- En el capítulo 2 se presenta la descripción del problema con los asistentes virtuales en la actualidad, así como la importancia del procesamiento del lenguaje natural aplicado a un asistente virtual. Se describe las diferentes medidas de similitud semántica presentes en la literatura.
- En el capítulo 3 describe las dos primeras fases de la metodología propuesta, la primera muestra los pasos para el procesamiento de un conjunto de datos, la segunda fase muestra la evaluación de 3 medidas de SST (Word2Vec,GloVe,FastFext), y por último demuestra cuál de las medidas anteriores da mayor resultado.
- En el capítulo 4 presenta la implementación de la 3 fase, que es aplicar el modelo de SST en un asistente virtual. Mostrando las evaluaciones que se obtuvieron en la aplicación del chatbot.
- En el capítulo 5 describe las conclusiones del presente documento, así como las limitaciones y el trabajo a futuro.

Capítulo 2

Estado del arte

En este capítulo se presenta la descripción del problema, la importancia del PLN, las medidas de similitud semántica textual y similitud semántica aplicada en los sistemas de conversación.

2.1. Descripción del problema

Un asistente virtual es un software de IA que puede simular una conversación (o un chat) con un usuario en lenguaje natural a través de aplicaciones de mensajería, sitios web, aplicaciones móviles o por teléfono. Se describe como una de las expresiones de interacción más avanzadas y prometedoras entre humanos y máquinas. Sin embargo, desde un punto de vista tecnológico, un asistente virtual representa la evolución natural de un sistema de respuesta a preguntas que aprovecha el PLN. La formulación de respuestas a preguntas en lenguaje natural es uno de los ejemplos más típicos de PLN aplicado en las aplicaciones de uso final de varias empresas, por ejemplo, gestión, personal, ventas, consultoría.

Las aplicaciones de los AV agilizan las interacciones entre personas y servicios, mejorando la experiencia del cliente. Al mismo tiempo, ofrecen a las empresas nuevas oportunidades para mejorar el proceso de participación de los clientes y la eficiencia operativa al reducir el costo típico del servicio al cliente.

El presente proyecto contribuirá en los asistentes virtuales proponiendo una

técnica de similitud semántica textual que ayudará a tener una mayor precisión de respuesta a las necesidades del usuario en un entorno conversacional.

2.2. Procesamiento de Lenguaje Natural

El lenguaje natural, es el lenguaje utilizado por el público en general para la comunicación diaria. Un lenguaje artificial se caracteriza por vocabularios de creación propia, gramática estricta y un rango ideográfico reducido y, por lo tanto, pertenece a una categoría lingüística. Un lenguaje natural varia constantemente con el tiempo, los individuos pueden desarrollar fácilmente un sentido de este primer idioma mientras crecen. Además, la flexibilidad sintáctica y semántica de un lenguaje natural permite que este tipo de lenguaje sea natural para los seres humanos.

El PLN estudia cómo permitir que una computadora procese y comprenda el lenguaje utilizado por los seres humanos en su vida diaria, para comprender el conocimiento humano y para comunicarse con los seres humanos en el lenguaje natural. Las aplicaciones de PLN incluyen Recuperación de Información (RI), extracción de conocimiento, sistemas de Preguntas y Respuestas(QA), categorización de texto, traducción automática, asistencia de escritura, identificación de voz, composición, entre otros. El desarrollo de Internet y la gran producción de documentos digitales han resultado en una necesidad urgente de procesamiento de texto inteligente, y la teoría, así como la habilidad de la PLN, se han vuelto más importantes.

La necesidad de un método de análisis semántico en documentos u oraciones más cortos se ha producido gradualmente en los campos de las aplicaciones de PLN en los últimos años.

La Similitud Semántica Textual(SST) tiene como objetivo capturar cuando el sentido de dos textos es similar. Este concepto difiere de encontrar el grado de similitud textual, que está solamente interesado en medir el número de componentes léxicas que comparten ambos textos. Encontrar la similitud semántica entre pares de oraciones y textos se ha convertido en un gran reto para los especialistas en PLN, ya que se puede aplicar en diferentes tareas de PLN mencionadas anteriormente.

Por otra parte, existen muchas maneras de evaluar el nivel de inteligencia de un Chatbot, uno de los aspectos más críticos es la capacidad de llevar a cabo conversaciones contextuales, atractivas y comprensibles con los seres humanos. Se trata del procesamiento de mensajes de texto enviado por un usuario, dividiéndolos en partes, agregando elementos gramaticales e identificando elementos importantes, por mencionar algunos como tokenización, normalización, etiquetado POS, N-gramas, entre otros.

2.2.1. La importancia del PLN en asistentes virtuales

Para que un AV pueda tener una conversación con un usuario, deben tener la capacidad del conocimiento sobre el contexto. Y para esa capacidad, es esencial entrenar AV con PLN. Con PLN, los AVs pueden entender fácilmente el complejo lenguaje humano. Cada persona posee un estilo diferente mientras se expresa. Con el PLN pueden captar rápidamente la personalidad de una persona y responder en consecuencia.

2.3. Similitud Semántica Textual

Las oraciones semánticamente relacionadas pueden no tener ninguna palabra en común. En el lenguaje natural una sola palabra puede tener más de un significado y diferentes palabras pueden compartir el mismo sentido (Polisemia y Sinónimo)[21]. Se asume que las oraciones con texto similar o palabras en común están relacionadas semánticamente. Si bien esta suposición es generalmente válida para documentos de texto, la suposición no se aplica a las oraciones, ya que dos oraciones pueden ser semánticamente similares a pesar de tener pocas palabras en común [10]. Esto significa que la medida estándar de RI basada en la co-ocurrencia de palabras no es apropiada para abordar los desafíos mencionados anteriormente de identificar semántica en un texto no estructurado.

Por otra parte, Blanco [4] presenta una herramienta que facilita el análisis semántico, la presentación y la comprensión, permite usar las matemáticas de consecuencia lógica para describir las relaciones semánticas entre oraciones y se usan para determinar la similitud textual. Se proponen tres transformaciones de forma lógica teniendo en cuenta la estructura semántica de las

oraciones. La primera transformación llamada Básico genera predicados para todos los nombres, verbos, modificadores y entidades nombradas, omite la estructura semántica de las oraciones y es equivalente a la contabilidad de las palabras de contenido, sus etiquetas de parte del discurso y los tipos de entidades nombradas. La segunda llamada SemRels genera predicados para todas las relaciones semánticas, conceptos que son argumentos de relaciones y entidades nombradas que califican esos conceptos, sino se encuentran relaciones semánticas en una oración, SemRels retrocede a Básico para evitar formas lógicas vacías. Y finalmente la tercera transformación llamado completo, genera predicados para todos los conceptos, todas las relaciones semánticas y todas las entidades nombradas, es equivalente a SemRels después de agregar predicados para conceptos que no son argumentos de una relación semántica, o alternativamente, equivalente a Básico después de agregar predicados para relaciones semánticas. Las pruebas lógicas se obtienen mediante un paso de resolución adaptado que elimina predicados cuando no se puede encontrar una prueba con resolución estándar. Varias medidas de similitud de palabra a palabra a menudo se combinan con otras características superficiales, por ejemplo, la superposición de n-gramas, dependencias sintácticas, utilizando el aprendizaje automático supervisado. Los resultados experimentales muestran que tener en cuenta las relaciones semánticas para determinar la similitud textual produce mejoras en el rendimiento con respecto al baseline y los sistemas de vanguardia de terceros, las puntuaciones WN sólo, que no tienen en cuenta la estructura semántica de las oraciones, ofrecen un baseline. Las puntuaciones WN utilizan las puntuaciones derivadas con las medidas de similitud de palabras por pares. Las relaciones semánticas son extraídas por Polaris, un analizador semántico que extrae relaciones de una amplia variedad de patrones sintácticos de léxico que incluyen estructuras verbo-argumentos, compuestos de sustantivos, genitivos y otros. Se entrena utilizando FrameNet, PropBank, NomBank, varias competiciones SemEval [12] y anotaciones internas. El enfoque para determinar la similitud textual se basa en características semánticas derivadas de un comprobador lógico que se combinan en un marco estándar de aprendizaje automático supervisado. La entrada es un par de oraciones: sent1 y sent2. La salida es una puntuación numérica que estima la similitud textual entre el par de oraciones. Primero, sent1 y sent2 se transforman en las formas lógicas lft1 y lft2. Luego, un modificador lógico modificado encuentra pruebas en ambas direcciones: de lft2 a lft2 y de lft2 a lft2. El promotor produce puntuaciones de similitud basadas en el número de predicados que se han eliminado (puntuaciones LFT) y características que

caracterizan las pruebas. Se obtienen puntuaciones de similitud adicionales con las medidas de similitud de palabras por pares estándar. Finalmente, todos los puntajes y características se combinan utilizando el aprendizaje automático supervisado para obtener el puntaje final.

Es importante mencionar que reconocer las oraciones de definición (oraciones que interpretan términos y conceptos) a partir de corpus de texto libre a menudo requiere patrones hechos a mano o instancias de entrenamiento explícitamente etiquetadas. Define una oración de definición como una que informa a los lectores "qué es (una) X", donde X es un término2. No se restringe a sentencias definitivas explícitas como "X se define como. . . ", aunque se cree que una solución efectiva debería reconocer tales oraciones como resultados de alta calidad.

La identificación de oraciones de definición a partir de textos es útil para muchas aplicaciones como la respuesta a preguntas [23], la construcción automática de tesauros, etc. Los primeros métodos desarrollan patrones léxico-sintácticos hechos a mano, como "X es (a) Y" para reconocer las oraciones de definición. Estos patrones tienen una efectividad limitada porque es difícil y lento enumerar todos los posibles patrones y excepciones. Los métodos más modernos [28, 14, 11] se basan en técnicas de aprendizaje automático supervisadas para identificar oraciones de definición. Estos métodos son más efectivos, pero también tienen dos limitaciones clave: costo de anotación (se requieren al menos miles de oraciones etiquetadas para entrenar modelos supervisados efectivos) y generalización (las oraciones etiquetadas y los modelos aprendidos son a menudo pertinentes a un corpus particular, que puede o no generalizarse bien a otros corpus). Para abordar estos desafíos, Jiang *et al.* [13] proponen un método de supervisión a distancia para generar juicios de oraciones de definición grandes y efectivos en corpus de nuevos textos que requieren poco esfuerzo de anotación manual. La supervisión a distancia (SD) se refiere a un paradigma de aprendizaje supervisado en el que los datos de entrenamiento no se anotan manualmente, sino que se generan automáticamente a partir de bases de conocimiento (BC) y heurísticas. Aplicaron este paradigma a la tarea. Primero se extrae un pequeño conjunto de oraciones de definición precisas de un (BC): en el caso, Wikipedia. Dos preocupaciones prácticas para los enfoques supervisados son el alto costo de recopilar juicios humanos y la generalización limitada a nuevos corpus. El método que proponen de supervisión distante basado en la similitud para abordar estos

problemas para la tarea de definición de recuperación de oraciones (dado un término objetivo X, recuperar y clasificar oraciones en un corpus de texto por su calidad de explicación de X). Los resultados experimentales verificaron algunas ventajas del método llamado supervisión a distancia basada en similitud Sim-DS: bajo costo, eficacia y flexibilidad. Define Sim-DS como: dada X con la etiqueta L (de una base de conocimientos), asigna L a Y (una entrada de datos en el objetivo del corpus) si X y Y son similares entre sí. El método Sim-DS para la definición de recuperación de oraciones es un problema Sim-DS donde: 1) tanto X como Y son oraciones; 2) X es una instancia positiva (en nuestro caso, una oración de definición). Demuestran que su método puede beneficiarse significativamente de las medidas de similitud de texto supervisadas aprendidas de datos de entrenamiento externos (de la tarea SemEval Similitud Semántica Textual) o locales (unos pocos cientos de oraciones evaluadas en el corpus objetivo), así como una amplia gama de aplicaciones, como los motores de búsqueda web y los sistemas de control de calidad.

2.4. Tareas de SemEval

SemEval es una serie continua de evaluaciones de sistemas de análisis semántico computacional. Las evaluaciones están destinadas a explorar la naturaleza del significado en el lenguaje. Esta serie de evaluaciones proporciona un mecanismo para caracterizar en términos más precisos exactamente lo que es necesario para calcular el significado.

SST mide la similitud de significado de las oraciones. La tarea de 2017 [8] se enfoca en pares multilingües e interlingual, está motivada por la observación de que modelar con precisión el significado de las oraciones es un problema fundamental de comprensión del lenguaje relevante para numerosas aplicaciones, entre ellas: Traducción Automática(TA), resumen, generación, respuesta a preguntas, calificación de respuesta corta, búsqueda semántica, dialogo y sistemas conversacionales.

SST permite la evaluación de técnicas de un conjunto diverso de dominios en comparación con un criterio de rendimiento interpretable compartido. Los conjuntos de datos de tareas compartidas de SST se han utilizado ampliamen-

te para la investigación sobre similitud a nivel de oraciones y representaciones semánticas.

La similitud textual puede ir desde la no relación completa hasta la equivalencia semántica exacta, y una similitud gradual capta intuitivamente la noción de matices intermedios de similitud, ya que los pares de texto pueden diferir de algunos aspectos matizados menores del significado, hasta diferencias semánticas relativamente importantes, hasta compartir sólo algunos aspectos, detalles, o simplemente por estar relacionado con el mismo tema.

Agirre *et al.* [1] desarrollan un marco unificado para combinar varios componentes semánticos que, de otro modo, han tendido a ser evaluados de forma independiente y sin caracterización del impacto en las aplicaciones del Procesamiento del Lenguaje. Natural PLN. Al proporcionar dicho marco, SST permite una evaluación extrínseca de estos módulos. Además, dicho marco de SST en sí mismo podría a su vez ser evaluado de forma intrínseca y extrínseca como una caja gris / negra dentro de varias aplicaciones del PLN, como la traducción automática, el resumen, la respuesta a preguntas, etc. consiste en el estilo multilingüe, al introducir conjuntos de datos en español junto con el inglés. Los conjuntos de datos que están en inglés intentan exponer a los equipos participantes a escenarios más diversos en comparación con los años anteriores, mediante la introducción de descripciones de imágenes, el foro y el género de noticias y los tweets de titulares de noticias. Para el español, desarrollan dos conjuntos de datos que consisten en texto enciclopédico y de noticias de última hora adquirido de fuentes españolas.

Por otra parte, Aldarmaki *et al.* [2] presenta un modelo de factorización matricial para el aprendizaje de representaciones lingüísticas cruzadas para oraciones. Usando corpus alineados con oraciones, el modelo propuesto aprende representaciones distribuidas al factorizar los datos dados en factores dependientes del idioma y un factor compartido. Como resultado, ingresa oraciones de ambos idiomas. Se puede mapear en vectores de longitud fija y luego comparar directamente utilizando la medida de similitud de coseno, que logra una correlación de Pearson de 0.8 en la SST español-inglés.

La investigación de Hershcovich *et al.* [24] presenta la tarea compartida SemEval 2019 sobre el análisis de la Anotación Cognitiva Conceptual Universal (UCCA) en inglés, alemán y francés. UCCA es un marco de referencia cru-

zada para la representación semántica, que se basa en un extenso trabajo tipológico y admite anotaciones rápidas. UCCA plantea un desafío para las técnicas de análisis existentes, ya que exhibe reentrada (lo que resulta en estructuras DAG), estructuras discontinuas y nodos no terminales correspondientes a unidades semánticas complejas. La tarea compartida ha producido mejoras sobre la línea de base de última generación en todos los idiomas y configuraciones. UCCA es un esquema de representación semántica aplicable en varios idiomas, que se basa en el marco tipológico establecido de la teoría lingüística básica (Dixon, 2010b, a, 2012). Ha demostrado su aplicabilidad a múltiples idiomas, incluidos inglés, francés y alemán, y los proyectos piloto de anotación se realizaron en algunos idiomas más. Se ha demostrado que las estructuras de UCCA están bien conservadas en la traducción (Sulem *et al.*, 2015), y que admiten la anotación rápida por parte de no expertos, con la ayuda de una interfaz de anotación accesible (Abend *et al.*, 2017). 1 UCCA ya ha mostrado un valor aplicativo para simplificación de texto (Sulem *et al.*, 2018b), así como para definir medidas de evaluación semántica para tareas de generación de texto a texto, incluida la traducción automática (Birch *et al.*, 2016), simplificación de texto (Sulem *et al.*, 2018a) y corrección gramatical de errores (Choshen y Abend, 2018). UCCA representa la semántica de las expresiones lingüísticas como gráficos acíclicos dirigidos (DAG), donde los nodos terminales (sin hijos) corresponden a los tokens de texto, y los nodos no terminales a unidades semánticas que participan en alguna relación súperordenada. Los bordes están etiquetados, lo que indica el papel de un niño en la relación que representa el padre. Los nodos y los bordes pertenecen a una de varias capas, cada una correspondiente a un "módulo"de distinciones semánticas.

2.5. Similitud semántica textual aplicada en sistemas de conversación

La mayoría de los motores de chatbot actuales están diseñados para responder a las declaraciones de los usuarios basándose en los pares de pregunta - respuesta (o QA) existentes.

Yan *et al.* [32], presentan DocChat, un novedoso enfoque de recuperación de información para los motores de chatbot que pueden aprovechar los documen-

tos no estructurados, en lugar de los pares Q-A, para responder a las preguntas. Proponen un modelo de aprendizaje para clasificar con características diseñadas a diferentes niveles de granularidad para medir la relevancia entre las declaraciones y las respuestas directamente. Evalúan su enfoque propuesto en inglés y chino: (i) Para inglés, evalúan DocChat en WikiQA y QASent, dos tareas de selección de oraciones de respuesta, y lo comparan con los métodos más modernos, los cuáles son (1) Yih *et al.* (2013), que hace uso de características semánticas léxicas; (2) Yang *et. al.* (2015), que utiliza un modelo de aprendizaje profundo Red Neuronal Convolucional (RNC) de dos gramos con agrupamiento promedio; (3) Miao *et. al.* (2015), que utiliza una red neuronal de Memoria a Corto y Largo Plazo (MCLP) enriquecido con un mecanismo de atención estocástica latente para modelar la similitud entre pares QA; y (4) Yin *et. al.* (2015), que agrega el mecanismo de atención a la arquitectura RNC. Se observan mejoras razonables y buena adaptabilidad. (ii) Para los chinos, comparan DocChat con XiaoIce2, un famoso motor chitchat en China, y la evaluación en paralelo muestra que DocChat es un complemento perfecto para los motores de chatbot que utilizan pares QA como fuente principal de respuestas.

Una investigación reciente de Luceri *et al.* [16] permitió conocer el problema de los bots en las redes sociales y los riesgos significativos de la manipulación masiva de la opinión pública en el contexto de la discusión política. En este trabajo, aprovecharon Twitter para estudiar el discurso durante las elecciones de 2018 en Estados Unidos y analizar la actividad de los bots sociales y las interacciones con los seres humanos. Recolectaron 2.6 millones de tweets durante 42 días alrededor del día de las elecciones de casi 1 millón de usuarios. Utilizan los tweets recopilados para responder a tres preguntas de investigación:

1. ¿Los bots sociales se inclinan y se comportan de acuerdo con una ideología política?

2. ¿Podemos observar diferentes estrategias entre los bots liberales y conservadores?

3. ¿Qué tan efectivas son las estrategias bot?

Muestran que los bots sociales pueden clasificarse con precisión de acuerdo con su tendencia política y comportarse en consecuencia. Los bots conservadores comparten la mayoría de los temas de discusión con sus homólogos

humanos, mientras que los bots liberales muestran menos superposición y una actitud más inflamatoria. Estudian las interacciones de bot con humanos y observan diferentes estrategias. Finalmente, miden la incrustación de los bots en la red social y la efectividad de sus actividades. Para este propósito,en términos de volumen y frecuencia de publicaciones, interacciones con los seres humanos e integración. Presentan 4 métricas para estimar la efectividad de los bots y, al mismo tiempo, calcularon en qué medida los humanos confían e interactúan con el contenido generado por los bots sociales, Retweet Pervasiveness (RTP) mide la intrusión de contenido generado por bot en retweets generados por humanos; Reply Rate (RR) mide el porcentaje de respuestas dadas por los humanos a los bots sociales; Human to Bot Rate (H2BR) cuantifica la interacción humana con bots sobre todas las actividades humanas en la red social y por último Tweet Success Rate (TSR) es el porcentaje de tweets generados por bots que obtuvieron al menos un retweet de un humano. Los resultados muestran que los bots conservadores están más profundamente integrados en la red social y más efectivos que los bots liberales al ejercer influencia sobre los humanos.

Presentan Athreya *et al.* [3] un chatbot DBpedia, basado en grafos de conocimiento diseñado para optimizar la interacción de la comunidad. El bot fue diseñado para la integración en el software de la comunidad para facilitar la respuesta a preguntas recurrentes. Al desarrollar el chatbot, se abordaron cuatro desafíos principales: (1) comprender las consultas de los usuarios, (2) obtener información relevante en función de las consultas, (3) adaptar las respuestas según los estándares de cada plataforma de salida (es decir, Web, Slack, Facebook)) y (4) el desarrollo de las interacciones subsiguientes del usuario con el chatbot de Dbpedia. Con esta demostración, mostraron soluciones a estos cuatro desafíos. Su arquitectura se basa en un enfoque modular para dar cuenta de los diferentes tipos de consultas y respuestas de los usuarios. El Chatbot de DBpedia es capaz de responder a los usuarios a través de (1) mensajes de texto cortos o simples (2) a través de elaborar mensajes interactivos utilizando el grafo de conocimiento subyacente. Los usuarios pueden comunicarse con DBpedia Chatbot a través de (3) texto, pero también a través de (4) interacciones, como hacer clic en botones o enlaces, así como dar comentarios. Además, se enfocan en diseñar este bot para que pueda conectarse a una multitud de plataformas para aumentar el alcance a la comunidad. El núcleo del DBpedia Chatbot se basa en Spring framework4 y puede ejecutarse en una sola máquina central o distribuirse en

un clúster. Los comentarios de los usuarios entrantes se almacenan en Couch DB de forma anónima para futuras investigaciones.

Khurana *et al.* [15] describen un asistente automatizado para responder preguntas frecuentes (FAQ) sobre consultas relacionadas con recursos humanos en dos áreas diferentes (gestión de licencias y seguro de salud). Las necesidades de una gran empresa global los lleva a modelar una FAQ para que sea una clase de equivalencia de preguntas realmente formuladas, para lo cual hay una respuesta común (certificada como consistente con la política de la organización). Cuando se plantea una nueva pregunta a la aplicación, encuentra la clase de pregunta y responde con la respuesta para la clase. Emplean una arquitectura híbrida de profundización en la que un clasificador basado en BiLSTM (Memoria bidireccional a corto y largo plazo) se combina con una segunda red Siamese basada en BiLSTM de manera iterativa: preguntas para las cuales el clasificador comete un error durante el entrenamiento se usan para generar un conjunto de pares preguntas y preguntas mal clasificadas que, junto con los pares correctos, se usan para entrenar a la red siamesa para separar las representaciones (ocultas) de los pares mal clasificados. BiLSTM son variantes de Redes Neuronales Recurrentes (RNNs), que son diseñados para mitigar el problema del gradiente oculto, que ocurre cuando los RNNs aprenden secuencias con patrones a largo plazo; y lo utilizan para el modelo de clasificación, es decir, la secuencia se da como entrada en orden directo e inverso, como resultado de cada palabra en la consulta que retiene el contexto de otras palabras tanto en el lado izquierdo como en el derecho. Desarrollan una arquitectura híbrida de aprendizaje profundo capacitada de forma iterativa, que combina la red de Siamese y clasificadores, o HSCM-IT para la tarea de responder preguntas. Un modelo Siamese, es una red capaz de modelar la similitud semántica subyacente de un par de oraciones de entrada [17], en su aplicación, la red toma muchos pares de consultas diferentes, xi + xj, algunas de las cuales pertenecen a la misma clase, mientras que otras pertenecen a diferentes clases, es decir, dado un par de consultas, el objetivo es predecir si pertenecen a la misma clase 1 o no 0. Las clases son el dataset HIS (Plan de licencia y seguro de salud) y Leave se organizan en varias categorías, por ejemplo, todas las clases relacionadas con la licencia por enfermedad se agrupan en la misma categoría, o todas las clases relacionadas con la de seguro de salud se agruparon en una categoría. HSCM-IT Capacitación Iterativa de Híbrido de modelos siamese y de clasificación, mejora la precisión de la clasificación, en cada iteración después de ejecutar el modelo de clasificación

identificamos los pares de consultas frecuentemente clasificadas erróneamente, es decir, si muchas consultas de una clase se predicen con frecuencia en otra clase en el conjunto de datos de validación.

Se sabe que los niños son interrogadores curiosos y persistentes. La omnipresencia de las interfaces de voz representa una oportunidad para que los niños que aún no son lectores con fluidez busquen en Internet de forma independiente haciendo preguntas a través de agentes de conversación como Amazon Alexa, Siri de Apple y el Asistente de Google. A través de un estudio B. Lovato *et al.* [22] de implementación en el hogar de dos semanas de duración que incluyó a 18 familias (niños de 5 a 6 años y sus padres), informan sobre las preguntas que los niños eligen hacerle al agente de conversación, las respuestas que el agente proporcionó, los desafíos en el uso y sus percepciones la tecnología. Con base a su análisis, identifican varias consideraciones para el diseño de agentes de conversación basados en la voz que tienen como objetivo apoyar el comportamiento de preguntas y el desarrollo posterior de los niños pequeños. Implementan altavoces inteligentes en hogares familiares, enfocándose en niños de 5 y 6 años, junto con un padre (N = 18 pares de niños y padres; 9 niños identifican como mujeres; edad promedio del niño = 5.97 años, DE = 0.64). Se centran en los niños de 5 y 6 años porque este es un período clave durante el cual emerge la alfabetización en la mayoría de los niños [21]. Realizan un estudio naturalista que incluye dos visitas domiciliarias con cada padre e hijo (juntos). Durante la primera visita, después de obtener el permiso y el consentimiento de los padres y el asentimiento del niño, se les pide a los padres que llenen un cuestionario con información demográfica y los niños completan la evaluación de lectura. Al final de la visita, se instala un dispositivo Google Home Mini en el hogar, vinculado a una cuenta única de Google accesible sólo para el equipo de investigación. Seleccionan Google Home Mini porque el dispositivo proporciona archivos de registro que incluyen archivos de sonido sin procesar y porque quieren evitar la complejidad adicional asociada con solicitudes de compra accidentales (es decir, Alexa podría malinterpretar una consulta e iniciar una serie de preguntas sobre cómo realizar una compra). Google Home Mini es un asistente de voz que ayuda a controlar la casa, organizarse y obtener información. Evita tener que el móvil para hacer cosas como poner música, la radio, buscar información básica o controlar aparatos de casa como la luz.

2.6. Medidas de Similitud Semántica

Word Embeddings

Es una técnica de modelado de lenguaje utilizada para mapear palabras a vectores de números reales. Representa palabras o frases en el espacio vectorial con varias dimensiones. Los word embeddings se pueden generar utilizando varios métodos como redes neuronales, matriz de co-ocurrencia, modelos probabilísticos, entre otros.

2.6.1. Word2vec

Es un grupo de modelos relacionados que se utilizan para producir word embeddings. Es tos modelos son redes neuronales superficiales de dos capas que están capacitadas para reconstruir contextos lingüísticos de palabras. Word2vec toma como entrada un corpus de texto y produce un espacio vectorial, de cientos de dimensiones, a cada palabras única en el corpus se le asigna un vector correspondiente. Los vectores de palabras se colocan en el espacio vectorial de tal manera que las palabras que comparten contextos comunes en el corpus se ubican cerca unas de otras en el espacio.

Ventajas

- La idea es muy intuitiva, que transforma el corpus sin etiquetar en datos etiquetados (asignando la palabra objetivo a su palabra de contexto) y aprende la representación de palabras en una tarea de clasificación.
- Los datos se pueden alimentar al modelo de forma en línea y necesita poco preprocesamiento, por lo que requieren poca memoria.
- El mapeo entre la palabra objetivo y su palabra de contexto incorpora implícitamente la relación sub-lineal en el espacio vectorial de las palabras, de modo que relaciones como “king:man y queen:woman”
- Es simple para un estudiante de primer año comprender el principio y hacer la implementación.

Desventajas

- las relaciones sub-lineales no están explícitamente definidas. Hay poco apoyo teórico detrás de tal característica.
- El modelo podría ser muy difícil de entrenar si usa la función softmax, ya que el número de categorías es demasiado grande(el tamaño del vocabulario). Aunque algoritmos de aproximación como muestreo negativo (NEG) y softmax jerárquico(HS).

Yang Lui *et. al.* [31] utiliza la combinación de similitud de palabras para estimar la similitud de oraciones. Emplean un modelo de regresión de vectores de soporte con características eficaces para predecir los puntajes de similitud entre pares de oraciones cortas en inglés. Los modelos que utilizan son características basadas en WordNet, Word2vec, alineación y basadas en literales para cubrir varios aspectos de la oración. El experimento que se realizó en la 2º Tarea de SemEval 2015 muestra que el método logró una correlación de Pearson: 80.486 %. Esto indica que la solución que emplearon es más competitiva cuando los datos de entrenamiento específicos del dominio están disponibles.
Word2vec consta de dos modelos para generar word embeddings [18]. El modelo bolsa continua de palabras (CBOW) y Skip Gram.

El modelo CBOW predice la probabilidad de una palabra dado su contexto. El contexto puede ser un número arbitrario de palabras. El número de palabras en el contexto viene dado por una ventana de contexto, que dice qué tan grande será el vecino de la palabra dada como contexto.

El modelo tiene una capa de entrada que contiene las palabras de contexto y la capa de salida contiene la palabra actual. La capa oculta contiene el número de dimensiones en la que queremos representar la palabra actual presente en la capa de salida. La representación de CBOW se puede ver en la figura 2.1

Tanto las palabras de contexto como la palabra de salida se representan en la red como vectores de tamaño únicos $|\mathcal{V}|$ (i.e. tamaño de vocabulario), indicando la posición de la palabra en el vocabulario.

Las palabras de contexto se introducen en la capa de entrada, y la red se entrena utilizando un algoritmo de retropropagación regular para generar la

palabra objetivo. Tenga en cuenta que cada palabra en el vocabulario tiene dos representaciones diferentes de dimensión igual al número de neuronas en la capa oculta. Una representación corresponde a las filas de la matriz de peso de entrada $\mathcal{W}$ y la segunda corresponde a las columnas de la matriz de peso de salida $\mathcal{W}$'. Por lo tanto, para una palabra ω,recibimos un vector de entrada $\mathbf{v}_w$ y un vector de salida $\mathbf{v}'_w$.

Hay una activación lineal en las neuronas de la capa oculta. Las neuronas de la capa de salida usan softmax como su activación para modelar la probabilidad de una palabra dado un contexto. Para una sola neurona en la capa de salida correspondiente a la palabra ω_O, obtenemos

$$p(\omega_O|\omega_I) = \sigma({\mathbf{v}'_{w_o}}^{\mathrm{T}}\mathbf{v}_{w_1}) = \frac{exp({\mathbf{v}'_{w_o}}^{\mathrm{T}}\mathbf{v}_{w_I})}{\sum w_j \in \mathcal{V} exp({\mathbf{v}'_{w_o}}^{\mathrm{T}}\mathbf{v}_{w_1})} \tag{2.1}$$

donde ω_I es una palabra de entrada(contexto). En un caso de contexto de varias palabras, el vector $\mathbf{v}_{w_1}$ se calcula como un promedio sobre todas las palabras de contexto.

El algoritmo de Word2Vec utiliza una función de pérdida de registro negativa como objetivo

$$L = -\log p(\omega_O|\omega_I). \tag{2.2}$$

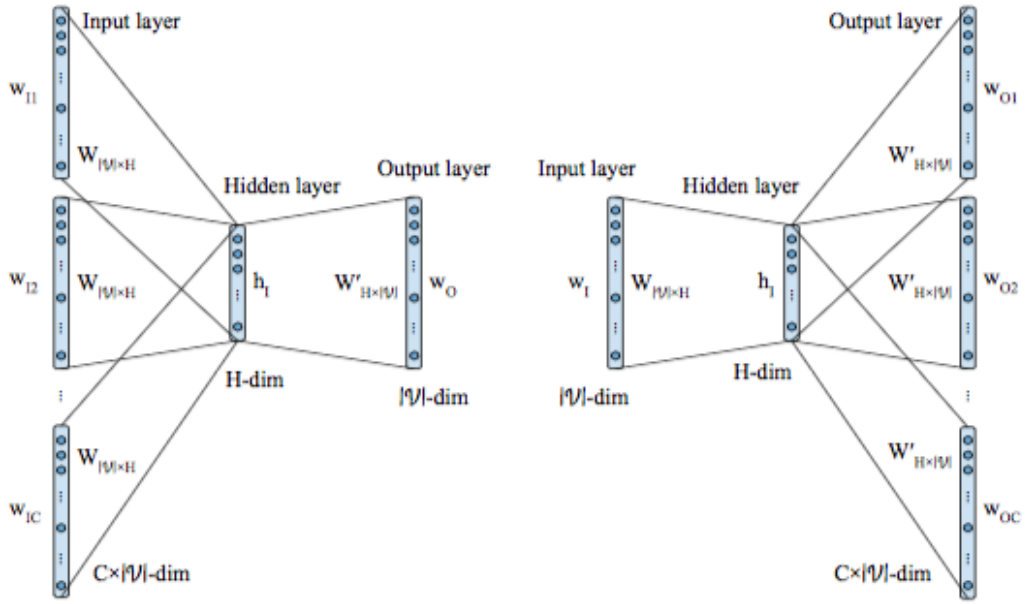

Figura 2.6.1: el esquema de una red neuronal general CBOW(izquierda) y Skip-Gram(derecha). $|\mathcal{V}|$ denota el tamaño del vocabulario, $\mathcal{C}$ es el tamaño de la ventana de contexto, $\mathcal{H}$ es la dimensión de incorporación, $\mathcal{W}$ y $\mathcal{W}'$ son matrices de entrada y salida ω_I y ω_O son las palabras de entrada y salida. Adaptado de [26].

La desventaja de CBOW es que, dado que utiliza el promedio del contexto de una palabra durante el cálculo de la activación oculta, no puede capturar múltiples significados semánticos de una sola palabra. El modelo Skip-Gram, aunque es más lento, resuelve este problema.

Como se muestra en la Figura 2.6.1, la arquitectura Skip-Gram es similar a la de CBOW, pero en lugar de intentar predecir una palabra dado su contexto, trata de predecir el contexto de una palabra. Durante el proceso de entrenamiento, habrá un error separado para cada palabra de contexto y esos vectores de error se agregarán en forma de elementos para obtener el error resultante, utilizando durante la propagación inversa.

La ventana de contexto funciona un poco diferente en el modelo de Skip-

Gram. En lugar de tener una ventana constante, se elige un rango de ventana máximo $\mathcal{C}$ y para cada palabra de entrenamiento, se selecciona aleatoriamente una ventana de contexto en el intervalo $[1, \mathcal{C}]$. Otra diferencia es que el modelo CBOW usa la matriz de salida $\mathcal{W}\prime$ como la incorporación de palabras resultante, mientras que el modelado Skip-GRam usa la matriz de entrada $\mathcal{W}$.

2.6.2. FastText

FastText (Bonjanowski *et. al.* [7]) propone un método para utilizar la información a nivel de caracteres. Es otro módulo para word embeddings y clasificar texto, ha sido desarrollado por Facebook y ha mostrado resultados en muchos problemas de PLN, como la detección de similitud semántica y la clasificación de texto.

Ventajas

- La estructura interna de la palabras al aprender representaciones de palabras, lo que podría ser muy para lenguajes morfológicamente, y también para las palabras que ocurren raramente.
- Puede generar word embeddings fuera de vocabulario(OOV), ya que aún puede crear embeddings como la suma de los n-gramas de caracteres presentes en la palabra desconocida.Bonjanowski *et. al.* [7] informan un rendimiento sorprendentemente bueno en tareas de similitud de palabras (OOV).
- Por la capacidad que tiene de crear word embeddings OOV, el algoritmo también es más robusto con respecto al tamaño de los datos de entrenamiento. Esto se puede usar para entrenar el algoritmo para tareas donde no hay grandes caorpus de entrenamiento.
- Entrenar un modelo FastText en el mismo tamaño de corpus que en el modelo Word2Vec será más lento, pero FastText puede lograr un rendimiento principalmente constante con el corpus de entrenamiento mucho más pequeño (o con menos epocas de entrenamiento) que Word2Vec.

Desventajas

- No es una representación distribuida: el número de dimensiones para cada vector crece con el tamaño del vocabulario. La matriz que se forma es muy escasa, lo que significa que la mayoría de los valores individuales son 0s. Por lo tanto, las manipulaciones de la matriz se vuelven computacionalmente demasiado caras incluso para un corpus de tamaño normal.

- Sin palabras de vocabulario:No es capaz de manejar nuevas palabras en el momento del examen.

- No es una representación distributiva: en la codificación de una sola vez, todos los vectores son equidistantes entre sí.

FastText es una extensión de Word2Vec. A diferencia de Word2Vec, que trata cada palabra como una entidad atómica, FastText representa cada palabra como una bolsa de carácteres n-gramas. Además de los n-gramas, la palabra misma se agrega a la bolsa para su propia representación. Cada palabra se cambia para incluir caracteres especiales <and>, que denotan el inicio y el final de la palabra. Es importante tener el cuenta que una palabra dada y un n-grama, que representa la misma secuencia de caracteres que esta palabra (por ejemplo, " <her> " y el carácter 3-gram "her"), tendrán representaciones vectoriales diferentes.

Softmax en la salida está modelado por la función de pérdida logistica $\ell(\mathcal{X}) = \log(1+e^{-x})$. El modelo de FastText es Skip-Gram (que ya incluye el muestreo negativo) para una muestra de entrenamiento se convierte en

$$L = \sum_{\omega_O \in \mathbf{C}_{\omega_I}} \ell(s(\omega_I, \omega_O)) + \sum_{\omega_n \in \mathbf{W}_{neg}} \ell(-s(\omega_I, \omega_n)), \tag{2.3}$$

donde $\mathcal{C}_\omega$ es el conjunto de palabras de contexto de la palabra ω, $\mathcal{W}_{neg}$ es el conjunto de muestras negativas actuales, y s (.,.) es una función de puntuación, que representa el puntaje de similitud entre las representaciones de dos palabras como

$$s(\omega, c) = \sum_{g \in \mathcal{G}_\omega} \mathbf{v}_g'^{\mathrm{T}} \mathbf{v}_c, \tag{2.4}$$

donde $\mathcal{G}_\omega \subset \mathcal{G}, \mathcal{G}$ es el vocabulario de n-gramas de caracteres, y $\mathbf{v}'_g$ es, por lo tanto, la representación vectorial de entrada de los n-gramas de caracteres presentes en la palabra ω. Este modelo permite compartir las representaciones vectoriales del carácter n-gramas entre las palabras. Luego se construye una palabra incrustada como la suma de sus propia representación y la representación de su carácter n-gramas.
En esta investigación Kaspars *et. al.* [5], utilizó FastText para generar word embeddings, ya que el uso que le da a las sub-palabras lo hace más apropiado para lenguajes con menos recursos y mayormente modificados. Evaluaron en 5 lenguajes conmúnmente utilizados en países bálticos. Asimismo, se exploraron métodos más sofisticados de normalización y autocorrección de errores para mejorar la exactitud de la detección de intentos.

2.6.3. StarSpace

La idea del algoritmo StarSpace [29] es codificar entidades de diferente tipos (por ejemplo, palabras, oraciones, documentos, etiquetas o clases de documentos, definiciones de usuario, etc.) es un espacio vectorial común. Cada entidad se describe mediante un conjunto de características del vocabulario (bolsa de características). El embedding de una entidad de características múltiples se construye como una suma de los embeddings de sus características. Las entidades se pueden comparar fácilmente mediante el uso de una función de similitud predefinida (por ejemplo, similitud de coseno) para resolver diversas tareas, clasificación, recomendaciones de documentos, búsqueda de artículos y similitud semántica. StarSpace utiliza la siguiente función como objetivo:

$$L = \sum_{\substack{(a,b)\in E^+ \\ b^-\in E^-}} L^{batch}(s(a,b), s(a,b_1^-), ..., s(a,b_k^-)), \tag{2.5}$$

donde E^+ es un generador específico de tareas de ejemplos positivos, E^- es un generador de ejemplos negativos que utiliza muestreo negativo, L^{batch} es una función de pérdida que compara el par positivo con los pares negativos y s(.,.) es una función de similitud, que luego se utiliza para comparar las entidades durante la prueba.

El algoritmo StarSpace implementa dos opciones para la función de pérdida L^{batch}. Cualquier pérdida logarítmica negativa de softmax, como se usa en los

algoritmos discutidos anteriormente, es decir $max(0, \mu - s(a, b))$, donde μ es un parámetro margen. La función de similitud también tiene dos opciones, ya sea un producto de puntos simple o la similitud de coseno como

$$s(a, b) = \frac{\mathbf{v}_a{}^{\mathrm{T}} \mathbf{v}_b}{||\mathbf{v}_a|| ||\mathbf{v}_b||}. \tag{2.6}$$

En el modo de aprendizaje no supervisado de word embeddings, se selecciona una palabra objetivo como la entidad b, y la entidad a representa las palabras en el contexto de la palabra objetivo.

Por lo tanto, si seleccionamos L^{batch} como la pérdida negativa logarítmica de softmax y la función de similitud s(.,.) como un producto de punto, la función objetivo resultante será igual al objetivo de FastText (2.3) sin considerar el carácter n-gramas.

2.6.4. GloVe

Los modelos actuales de word embeddings generalmente se entrenan mediante la factorización matricial de co-ocurrencia o por una red neuronal que utiliza ventanas de contexto local (Word2Vec, FastText, entre otros). Los modelos de factorización matricial utilizan los datos estadísticos del corpus, pero tienden a fallar al capturar la semántica de las palabras. Por otro lado, los métodos de la ventana de contexto son buenos para capturar el significado, sin utilizar los datos estadísticos. el algoritmo GloVe (vectores globales)(Pennington *et. al.* [20]) muestra que es posible usar métodos de factorización de matriz para mantener los datos estadísticos de co-ocurrencia mientras se puede capturar el significado de las palabras.

Ventajas

- El objetivo de GloVe es muy sencillo, es decir, hacer cumplir los vectores de palabras para capturar relaciones sub-lineales en el espacio vectorial. Por lo tanto, demuestra un mejor desempeño que Word2Vec en las tareas de analogía de palabras.
- Agrega un significado más práctico a los vectores de palabras al considerar las relaciones entre el par de palabras y el par de palabras en lugar de la palabra y la palabra

- Da menos peso a los pares de palabras muy frecuentes para evitar que las palabras de parada sin sentido como "the","an"no dominen el progreso del entrenamiento.

Desventejas

- El modelo está entrenado en la matriz de co-ocurrencia de palabras, que requiere mucha memoria para el almacenamiento. Especialmente, si cambia los hiperparámetros relacionados con la matriz de co.ocurrencia, debe reconstruir la matriz nuevamente, lo que consume mucho tiempo.

GloVe puede capturar las relaciones entre palabras mediante el uso de las proporciones de sus coincidencias con otras palabras ($P_{\omega_i\omega_k}/P_{\omega_j\omega_k}$). La probabilidad de co-ocurrencia se define como

$$\mathbf{P}_{\omega_i\omega_k} = \frac{\mathbf{X}_{\omega_i\omega_k}}{\sum_{\omega_k\in\mathcal{V}} X_{\omega_i\omega_k}}, \tag{2.7}$$

donde $X_{\omega_i\omega_k}$ es el número de veces que la palabra ω_k aparece en el contexto de la palabra ω_i, extraída de la matriz de co-ocurrencia X.

Para las palabras ω_k relacionadas con la palabra ω_i, podemos esperar una relación alta y para las palabras ω_k relacionadas con la palabra ω_j, podemos esperar que la relación sea pequeña. Para palabras completamente no relacionadas o relacionadas con ω_i y ω_k, la relación debe ser cercana a uno.

El algoritmo crea dos matrices de vectores de palabras, W y W'. Estas dos matrices son equivalentes en caso de que X sea simétrica y difiera sólo debido a su inicialización aleatoria. Pennington *et. al.* [20] afirman que entrenar múltiples instancias de una red neuronal y luego combinar los resultados puede ser menos propenso al sobreajuste y puede mejorar el rendimiento para ciertos tipos de redes neuronales. Aunque GloVe utiliza la factorización matricial en lugar de un modelo de red neuronal, este enfoque se adopta aquí. Por lo tanto, las incorporaciones resultantes se construyen sumando las matrices W W' y los vectores $\mathbf{v}_\omega$ y $\mathbf{v}'_\omega$ corresponden a estas dos matrices, en contraste con los algoritmos discutidos anteriormente.

GloVe propone un modelo de regresión ponderado de mínimos cuadrados como

$$J = \sum_{\omega_i \omega_j \in \mathcal{V}} r(\mathbf{X}_{\omega_i \omega_j})(\mathbf{v}_{\omega_i}{}^{\mathrm{T}} \mathbf{v}'_{\omega_j} + b_{w_i} + b'_{w_j} - \log X_{w_i w_j})^2, \tag{2.8}$$

donde b_ω y b'_ω son sesgos para $\mathbf{v}_\omega$ y $\mathbf{v}'_\omega$, y $r(\bullet)$ es una función de ponderación.

$$r(x) = \begin{cases} (x/x_{max})^\alpha & \text{si } x < x_{max}; \\ 1 & \textit{de otra manera}, \end{cases} \tag{2.9}$$

donde x_{max} denota eñ recuento de co-ocurrencia, después de lo cual el peso ya no aumentará. Esto se hace para no tener frecuentes apariciones de sobrepeso.Pennington *et. al.* [20] empíricamente encontrado $\alpha = 3/4$ a rendir al máximo en sus modelos.

2.6.5. Sent2Vec

Sent2Vec presenta un objetivo simple pero eficiente sin supervisión para entrenar representaciones distribuidas de oraciones. Puede considerarse como una extensión de Word2Vec (CBOW)(Pagliardini *et. al.*, 2017 [19]) a las oraciones, donde los vectores de palabras se optimizan para construir posteriormente embeddings de oraciones.

Embeddings de oraciones se define como un promedio simple sobre todas las palabras en la oración como

$$\mathbf{v}_S = \frac{1}{|R(S)|} \sum_{\omega \in R(S)} \mathbf{v}_\omega, \tag{2.10}$$

donde R(S) es el conjunto de palabras n-gramas presentes en la oración S(es decir, el conjunto de palabras en una oración en caso de que sólo se consideren unigramas).

La función objetivo utiliza la salida softmax aproximada por muestreo negativo. El objetivo de entrenamiento no supervisado para una oración de entrenamiento S se formula como

$$L = \sum_{\omega_O \in \mathcal{S}} \ell(\mathbf{v}_{w_O}{}^{\mathrm{T}} \mathbf{v}'_{S \setminus \{\omega_O\}}) + \sum_{\omega_n \in \mathcal{W}_{neg}} \ell(-\mathbf{v}_{\omega_n}{}^{\mathrm{T}} \mathbf{v}'_{S \setminus \{\omega_O\}}). \tag{2.11}$$

2.6.6. InferSent

El algoritmo InferSent explota word embeddings previamente entrenadas en grandes cuerpos de manera no supervisada (por ejemplo, Conneau *et. al.* [9] usan los vectores GloVe disponibles públicamente) mientras construye un clasificador supervisado en la parte superior. El clasificador esta entrenado en el corpus SNLI(Bowman *et. al.* [6]), que es una gran colección de pares de oraciones escrita por humanos (es decir, la premisa y los pares de hipótesis), etiquetados manualmente para la tarea de inferencia del lenguaje natural (ILN), es decir, reconocimiento de la vinculación textual - clases "vinculación", "neutral" y "contradicción".

El modelo InferSent consta de dos partes:

1. Codificador de oraciones que toma vectores de palabras y codifica oraciones en vectores.

2. Un clasificador (ILN) que toma los vectores codificados y genera una clase entre implicación, contradicción y neutral.

Codificador de oraciones

El modelo está entrenado en una manera que separa la codificación de las oraciones en los pares de oraciones. Se utiliza un codificador de oraciones compartidas, que genera un vector para la premisa $\mathbf{u}$ y la hipótesis $\mathbf{v}$. Después de eso, se utiliza tres métodos para extraer relaciones entre la premisa y la hipótesis:concatenación $(\mathbf{u}, \mathbf{v})$, producto de elementos $\mathbf{u} \odot \mathbf{v}$ y diferencia absoluta $|\mathbf{u} - \mathbf{v}|$. El vector resultante $(\mathbf{u}, \mathbf{v}, |\mathbf{u} - \mathbf{v}|, \mathbf{u} \odot \mathbf{v})$ se utiliza como entrada para un clasificador de 3 clases con una capa de salida softmax, que está entrenado para generar las clases definidas en el conjunto de datos SNLI (cf. Figura 2.6.6)

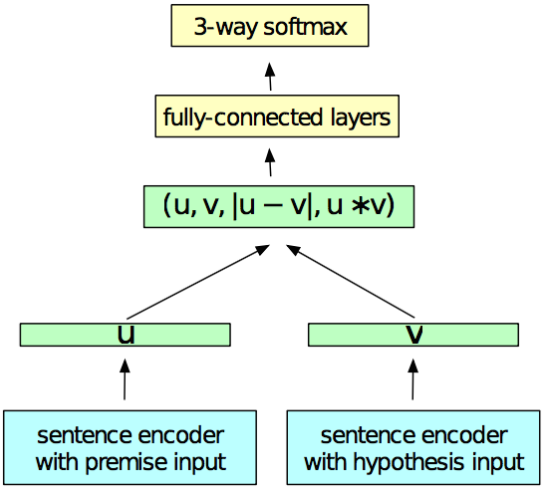

Figura 2.6.6: esquema de entrenamiento de Inferencia de Lenguaje Natural(ILN). Las oraciones de premisa e hipótesis se toman del SNLI Corpus antes de codificarlas en los vectores **u** y **v**, respectivamente. Las combinaciones de estos vectores se alimentan luego a un clasificador de 3 clases(por ejemplo, red de perceptrones múltiples capas) que está entrenado para generar clases de " vinculación ", " neutral" y " contradicción ". Adaptado de [25].

Conneau *et. al.* [9] utiliza un perceptrón de múltiples capas con una sola capa oculta de 512 neuronas como clasificador mientras experimenta con diferentes arquitecturas para el codificador de oraciones compartido. Además, concluyen que una red LSTM bidireccional con agrupación máxima supera a otras arquitecturas de red. El esquema de red propuesto se puede ver en la Figura 2.3.

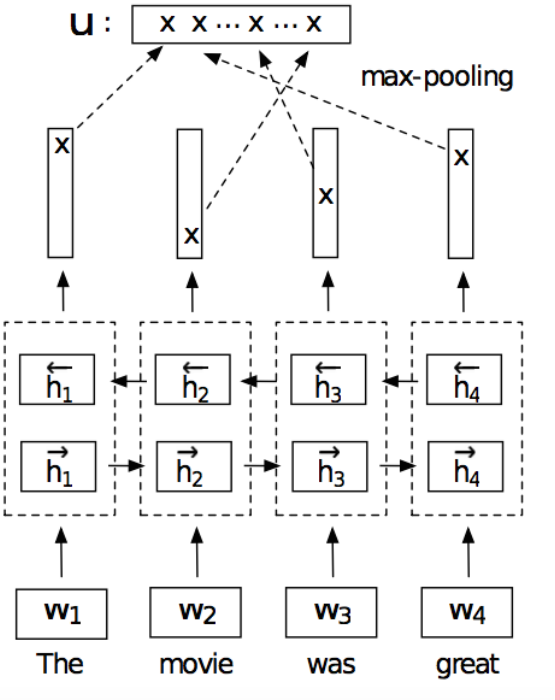

Figura 2.6.6.1: red LSTM bidireccional con esquema de agrupación máxima utilizada como codificador de oraciones en InferSent.Una oración se alimenta como una secuencia de palabras **T** en un LSTM hacia adelante y hacia atrás, que lee las oraciones en dos direcciones opuestas. Para $\mathbf{t} \in [1, ..., T]$,el vector $\mathbf{h}_t$ es la concatenación de las representaciones ocultas en la **t**-ésima capa de las dos redes.El concepto de agrupación máxima (es decir, seleccionar el valor máximo sobre cada dimensión de las unidades ocultas) se utiliza para crear un vector de tamaño fijo como una representación de oración. Adaptado de [25].

2.6.7. SoftMax

La mayoría de los algoritmos discutidos usan la función softmax en una salida para modelas la probabilidad de una palabra en un contexto dado. Todos estos modelos necesitan actualizar tanto los pesos de entrada **W** como los pesos de salida $\mathbf{W}'$ para cada muestra de entrenamiento. La actualización de la matriz de peso de entrada es barata ya que utiliza la activación lineal, pero la actualización de la función de softmax para la matriz de peso de

salida es muy costosa computacionalmente. La complejidad de actualizar un vector de palabra de salida $\mathbf{v}'_\omega$ para cada muestra de entrenamiento es $\mathcal{O}(|\mathcal{V}|)$, cuando se usa la implementación ingenua de softmax. Esto hace que la capacitación sea inviable para grandes corporaciones de capacitación. Las siguientes aproximaciones resuelven este problema limitado el número de operaciones requeridas.

Softmax jerárquico

Softmax jerárquico utiliza un árbol binario, donde cada palabra en el vocabulario se representa como una hoja (véase la Figura 2.6.7.1). Se puede demostrar que tal árbol tendrá $|\mathcal{V}|$ nodos internos.

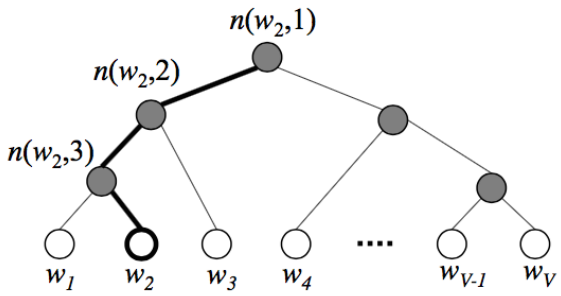

Figura 2.6.7.1: ejemplo jerárquico de árbol binario softmax. Las unidades blancas representan los nodos de la hoja(es decir, palabras en el vocabulario) y las unidades oscuras representan los nodos internos. Adaptado de [30].

En el modelo jerárquico softmax, los vectores de las palabras de salida $\mathbf{v}'_\omega$ se reemplazan con representaciones vectoriales de $|\mathcal{V}|-1$ nodos internos $\mathbf{v}'_{n(\omega,d)}$, representa el nodo $\mathbf{n}(\omega, d)$ a profundidad d en el camino desde la raíz hasta la palabra ω. La probabilidad de que una palabra sea la palabra de salida se puede calcular siguiendo una ruta desde la raíz hasta la hoja correspondiente como

$$p(\omega = \omega_O) = \prod_{d=1}^{D(\omega)-1} \sigma([\![n(\omega, d+1) = l(n(\omega, d))]\!] \bullet \mathbf{v}'_{n(\omega,d)}{}^{\mathrm{T}} \mathbf{v}'_{\omega_I}), \qquad (2.12)$$

donde $l(n)$ es el hijo izquierdo del nodo n, y

$$[\![x]\!] = \begin{cases} 1 & \text{si } x < \textit{verdadero}; \\ -1 & \textit{de otra manera}, \end{cases} \tag{2.13}$$

Por lo tanto, la probabilidad de que una palabra sea la palabra de salida se define como la probabilidad de una caminata aleatoria desde la raíz hasta la hoja correspondiente, donde la probabilidad de ir hacia la izquierda o hace la derecha en cada nodo interno viene dada por $\sigma([\![\bullet]\!] \bullet \mathbf{v}'^{\mathrm{T}}_n \mathbf{v}_{\omega_I})$. Por lo tanto, la probabilidad de la palabra ω_2 del ejemplo de la Figura 2.4 se puede calcular como

$$p(\omega_2 = \omega_O) = \sigma(\mathbf{v}'_{n(\omega_2,1)} \bullet \sigma(\mathbf{v}'^{\mathrm{T}}_{n(\omega_2,2)} \mathbf{v}_{\omega_I}) \bullet \sigma(\text{-}\mathbf{v}'^{\mathrm{T}}_{n(\omega_2,3)} \mathbf{v}_{\omega_I})) \tag{2.14}$$

Gracias a este enfoque, la probabilidad de un único resultado $p(\omega = \omega_O)$ sólo depende de los nodos internos que se encuentran en el camino desde la raíz hasta la hoja que denota ω, reduciendo efectivamente la complejidad de calcular el softmax desde $\mathcal{O}(|\mathcal{V}|)$ a $\mathcal{O}(\log_2 |\mathcal{V}|)$. Explorar el árbol con una búsqueda en profundidad también permite descartar las ramas con una pequeña probabilidad.

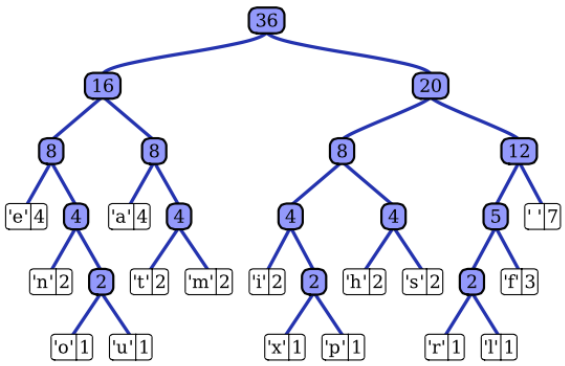

Figura 2.6.7.2: ejemplo de Wikipedia de un árbol Huffman construido a partir de caracteres en el oración "este es un ejemplo de un árbol Huffman". Adaptado de [27]

En cuanto al árbol en sí, el enfoque habitual en los algoritmos de embeddings es crear un árbol Huffman (véase en la Figura 2.5), que minimiza la longitud promedio del camino desde la raíz hasta la hoja.

Capítulo 3

Método

En el capítulo 2 se describió en el estado del arte diferentes medidas de SST presentes en la literatura, partiendo de esto, se presenta en este capítulo la evaluación de 3 medidas de SST para poder elegir que medida da mayor resultado y aplicarlo en un chatbot de preguntas frecuentes.

3.1. Introducción

Este capítulo explica 3 medidas de SST, 1) Word2Vec creado por Tomas Mikolov en google (2013), 2) Glove creado en Stanford (2014) y 3) FastText creado por Facebook (2018). La primer medida usa un modelo de red neuronal para aprender asociaciones de palabras en un dataset, utilizando un método para construir embeddings. Se puede obtener usando dos métodos (ambos con redes neuronales): Skip Gram y CBOW. La segunda medida es un algoritmo de aprendizaje no supervisado para obtener representaciones vectoriales de palabras. Se realiza un entrenamiento en estadísticas globales de co-ocurrencia palabra-palabra agregadas de un corpus, y los resultados muestran subestructuras lineales del espacio vectorial de palabras. La tercer medida es un aprendizaje eficiente de embeddigns de palabras y clasificación de oraciones, se pueden utilizar para numerosas aplicaciones, desde la comprensión de datos, como características en modelos adicionales, para la selección de candidatos o como inicializadores para el aprendizaje de transferencia. Al igual que Word2Vec, utiliza dos métodos Skip Gram y CBOW. En la fase 1 se expone el procesamiento de los documentos de texto, eliminando aquellos elementos innecesarios. La fase 2 se explica las medidas de SST eva-

luadas para obtener mayor precisión en cada oración. Por último en la fase 3 se muestra la técnica de SST que mayor acierto obtuvo con un conjunto de datos.

3.2. Fase 1 "Procesamiento del dataset"

En esta fase incluye la preparación de datos y limpieza, cuyo objetivo es reducir la complejidad de los mismos, detectando o eliminando elementos considerados superfluo, por ejemplo caracteres especiales, contracciones, entre otros. Para llevar a cabo dicho procesamiento se realizan los siguientes pasos que se muestra en la figura 3.1: 1) se eliminan los símbolos (?, $, %,&, %,!,?,œ, etc); 2) reemplazar las contracciones con sus expansiones (what's - what is); 3) reemplazar las palabras a minúsculas (What - what) y por último 4) separar las preguntas con las respuestas.

Algorithm 1 Preproceso Dataset

Require: datasetPreguntasFrecuentes
Ensure: Conjunto de preguntas frecuentes

```
data = file.readlines()
for line in data do
    eliminarSimbolos = re.sub(r'[?j$j %j" j(j)j&jœj!j]', r'', line):
    reemplazarContracciones = contractions.fix(eliminarSimbolos);
    convertirAMinusculas = reemplazarContracciones.lower();
    oracion = convertirAMinusculas.split(" \ t");
end for
return datosProcesados
```

3.3. Fase 2 "Evaluación de medidas de SST"

Para esta fase, se evaluaron 3 medidas de similitud, las cuales son: Word2Vec (2013), GloVe (2014) y FastText (2018).

Word2Vec

Word2Vec, tal como lo define Tensorflow, es un modelo que se utiliza para las representaciones vectoriales de palabras, llamada "word embeddings" creado por Mikolov et at. Hay 3 bloques de construcción principales que componen Word2Vec: 1) constructor de vocabulario, 2) contructor de contexto y 3) red neuronal con dos capas.

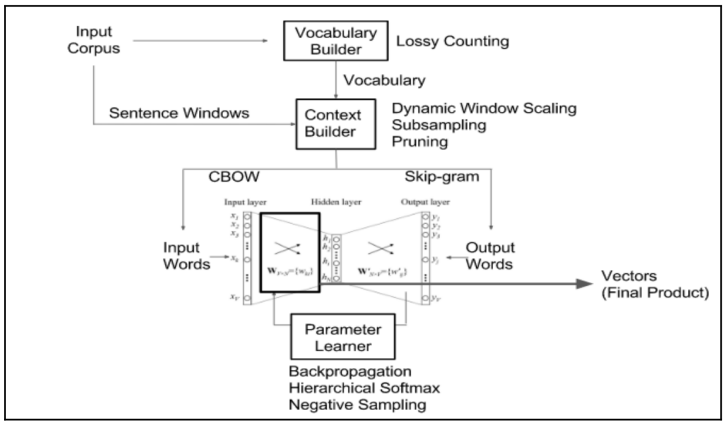

Figura 3.3.1: Arquitectura de Word2Vec.

Constructor de vocabulario

Este módulo es el componente básico de Word2Vec. Toma los datos sin antes procesarlos en forma de oraciones y extrae palabras únicas de estos para construir el vocabulario del sistema.

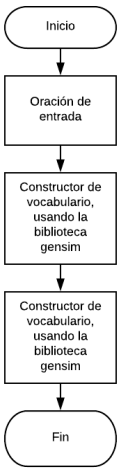

Figura 3.3.2: Constructor de vocabulario.

Constructor de contexto

Este es el siguiente paso para convertir palabras en vectores. La salida del constructor de vocabulario, es decir, el objeto de vocabulario que contiene el índice y el recuento, como entrada. No sólo toma la palabra en particular sino todas las palabras en la ventana de rango de contexto. La ventana de contexto es el número de palabras que debe tenerse en cuenta al recibir la entrada. Por lo regular, se consideran estándar de cinco a diez palabras.

Figura 3.3.3: Palabras en la ventana de rango de contexto.

Si se usa 5, entonces se toman en consideración 5 palabras desde la palabra central hacia la izquierda y la derecha. Esta forma los pares de palabras, el centro y las palabras de la ventana de contexto, que se alimentan a la red neuronal. Esto aumenta el contexto de la palabra central con los pares, lo que ayuda a definir el significado revelante de la palabra. Entonces, este par de palabras es la salida del generador de contexto y pasará al siguiente componente, que es una red neuronal de dos capas.

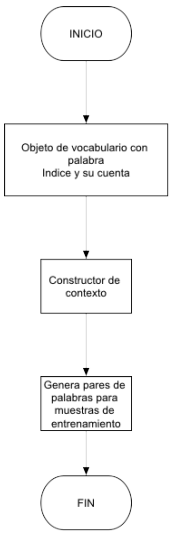

Figura 3.3.4: Constructor de contexto.

Red neuronal con dos capas

Word2Vec utiliza la red neuronal para el entrenamiento. Existe las siguientes capas: hay una **capa de entrada** que tiene tantas neuronas como palabras en el vocabulario para el entrenamiento. La segunda capa es la **capa oculta**, el tamaño de la capa en términos de neuronas es la dimensionalidad de los vectores de las palabras resultantes. La tercera y última capa es la **capa de salida** que tiene el mismo número de neuronas que la capa de entrada.

Codificación One-Hot

La entrada de la red neuronal será una palabra representada como un **one-hot vector**, es decir, un vector con tantas posiciones como un tamaño tenga el vocabulario. Por ejemplo, si quremos representar la palabra "ants" de un vocabulario 10.000 palabras, usaremos un vector de esa dimensión con 0 en todas las posiciones menos la correspondiente a la palabra "ants" que tendrá un 1, como se muestra en la figura 3.3.5.
La salida de la red neuronal será otro vector de las mismas 10.000 posiciones que representará las probabilidades de cada una de las palabras sean vecinas de la palabra representada en la entrada.

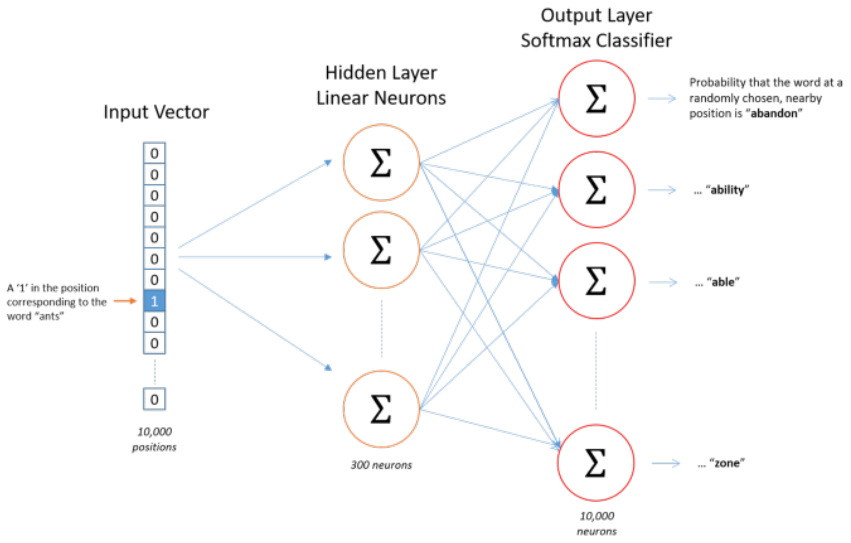

Figura 3.3.5: Representación de la arquitectura de la red neuronal.

Capa oculta

La capa oculta no tiene función de activación y en cambio la de salida aplica una función softmax. Aunque en el entrenamiento, tanto la entrada y como el valor esperado son **one-hot vector**, la salida es en realidad una distribución de probabilidades.
En el diagrama anterior, se está entrenando vectores de palabras con 300 características. La matriz de representación tendrá por tanto 10.000 filas (una por palabra) y 300 columnas (una por neurona).

Capa de salida

Para que la capa de salida cumpla con lo especificado anteriormente, se usa un clasificador softmax, hace que todos los valores estén entre 0 y 1 y que la suma de todos los valores sea 1, es decir, que sea una distribución de probabilidades.

Similitud de coseno

Los word embeddings generadas deben compararse para obtener similitudes semánticas entre dos vectores. Se utilizan pocos métodos estadísticos para encontrar la similitud entre dos vectores como: similitud de coseno, distancia del motor de palabras y distancia euclediana. Los modelos Word2Vec, FastText y GloVe, para obtener la similitud utilizan la similitud de coseno.
Similitud de coseno es la medida de la similitud existente entre dos vectores en un espacio que posee un prodcuto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. Esta función trigonométrica proporciona un valor igual a 1 si el ángulo comprendido es cero, es decir si ambos vectores, el coseno arrojaría un valor inferior a uno. Si los vectores fuesen ortogonales el coseno se anularía, y si apuntanse en sentido contrario su valor sería -1. De esta forma, el valor de esta métrica se encuentra entre -1 y 1, s decir en el intervalo cerrado[-1,1].
La similitud coseno de dos vectores n-dimensionales A y B se define como:

$$\frac{A \cdot B}{\parallel A \parallel \parallel B \parallel} = \frac{\sum_{n=1}^{N} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{3.1}$$

que simplemente es el coseno del ángulo entre A y B.

3.3.1. Algoritmo de Word2Vec

Como ya se menciono en el estado del arte, hay dos principales algoritmos de entrenamiento para Word2Vec, uno es la bolsa continua de palabras (CBOW), y el segundo es llamado Skip-gram. La principal diferencia entre estos dos métodos es que CBOW utiliza el contexto para predecir una palabra objetivo, mientras que skip-gram utiliza una palabra para predecir un contexto objetivo. Generalmente, el método skip-gram puede tener un mejor rendimiento en comparación con el método CBOW, ya que puede capturar dos semánticas para una sola palabra.

Biblioteca Gensim Python

Gensim es una biblioteca de Python de código abierto para el PLN, desarrollado por Radim Řehůřek. La biblioteca Gensim permite desarrollar **word embeddings** entrenando el propio modelo Word2Vec en el corpus personalizado. Se requiere instalar el paquete gensim. Se ejecuta en Linux, Windows

y Mac OS X, en cualquier otra plataforma que admita Python 2.7+ y NumPy.

Gensim depende del siguiente software:

- Python = 2.7
- Numpy = 1.11.3
- SciPy = 0.18.1

Hay dos formas de instalación:

1. pip install –upgrade gensim
2. conda install -c conda-forge gensim

Word2Vec con gensim
El Algoritmo 2 lleva a cabo el proceso de obtener la similitud entre una oración y otra. Tal algoritmo recibe como entrada un conjunto de datos que contenga preguntas, cuyo objetivo será procesar la información para poder obtener como resultado la similitud. Luego se crea el modelo skip-gram con los datos ya procesados, donde *min-count* es el recuento mínimo de palabras a considerar al entrenar el modelo, las palabras con ocurrencia menor que este recuento serán ignoradas, el valor predeterminado es 5; *size* es el número de dimensiones de las incrustaciones y el valor predeterminado es 100; *window* la distancia máxima entre una palabra de destino y palabras alrededor de la palabra de destino, la ventana predeterminada es 5; *sg* el algoritmo de entrenamiento para skip-gram es de 1. Con este modelo se obtiene la similitud semántica de cada palabra, para saber con exactitud la similitud por oración, se guardan los valores por cada palabra y al finalizar el recorrido de la oración se divide entre el total de palabras para así dar el resultado preciso de SST.

Algorithm 2: Word2Vec Skip-GRam

```
Input: datasetPreguntasFrecuentes
Output: similitud de pregunta1 a preguntaN
datosProcesados = procesar(corpusPreguntasFrecuentes)
SkipGram = gensim.models.Word2Vec(datosProcesados, min-count =
  1, size = 100, window = 5, sg = 1)
for key in sentences do
    for s1 in preguntaUsr do
        for s2 in preguntaDoc do
            sumaSkipGram = SkipGram.similarity(s1.lower(), s2.lower())
             count +=1
            guardaSumaSkipGram(sumaSkipGram)
        /* Devuelve la similitud de la pregunta deel usuario y la pregunta
           del corpus                                                    */
        resultadoSimilitud = ( suma(guardaSumaSkipGram) / count)
        /* Se limpia la lista dónde se almacena la similitud de la palabra
           s1 y palabra s2                                               */
        clearguardaSumaSkipGram
```

A diferencia del Algoritmo 2, el Algoritmo 3 obtiene la SST entre una oración y otra con el modelo CBOW. Toma un corpus de preguntas, después se crea el modelo CBOW con los datos ya procesados, donde *min-count* es el recuento mínimo de palabras a considerar al entrenar el modelo, las palabras con ocurrencia menor que este recuento serán ignoradas, el valor predeterminado es 5; *size* es el número de dimensiones de las incrustaciones y el valor predeterminado es 100; *window* la distancia máxima entre una palabra de destino y palabras alrededor de la palabra de destino, la ventana predeterminada es 5. Con este modelo se obtiene la SST por cada palabra, para saber con exactitud la similitud por oración, se guardan los valores por cada palabra y al finalizar el recorrido de la oración se divide entre el total de palabras para así dar el resultado preciso de SST.

Algorithm 3: Word2Vec CBOW

```
Input: datasetPreguntasFrecuentes
Output: similitud de pregunta1 a preguntaN
datosProcesados = procesar(corpusPreguntasFrecuentes)
CBOW = gensim.models.Word2Vec(datosProcesados, min-count = 1,
 size = 100, window = 5)
for key in sentences do
    for s1 in preguntaUsr do
        for s2 in preguntaDoc do
            sumaCBOW = CBOW.similarity(s1.lower(), s2.lower())
             count +=1
            guardaSumaCBOW(sumaCBOW)
        /* Devuelve la similitud de la pregunta deel usuario y la pregunta
           del corpus                                                    */
        resultadoSimilitud = ( suma(guardaSumaCBOW) / count)
        /* Se limpia la lista dónde se almacena la similitud de la palabra
           s1 y palabra s2                                               */
        clearguardaSumaCBOW
```

Resultados

Aplicando los dos algoritmos anteriores con un conjunto de datos obtenido por amazon en el apartado de electrónica, donde se obtuvieron 150 preguntas frecuentes. A cada algoritmo se le aplico dos fórmulas, la primera obteniendo el promedio y la segunda el máximo, para comparar los resultados y ver con que fórmula se daba una mayor similitud.

$$Promedio = \sum_{i=1}^{k} \sum_{j=1}^{r} similitudSemantica(wi, pj), \tag{3.2}$$

$$Max = \sum_{i=1}^{k} \sum_{j=1}^{r} similitudSemantica(wi, pj). \tag{3.3}$$

A continuación se muestran los resultados de la similitud obtenida por algoritmo Word2Vec con la arquitectura Skip-Gram.

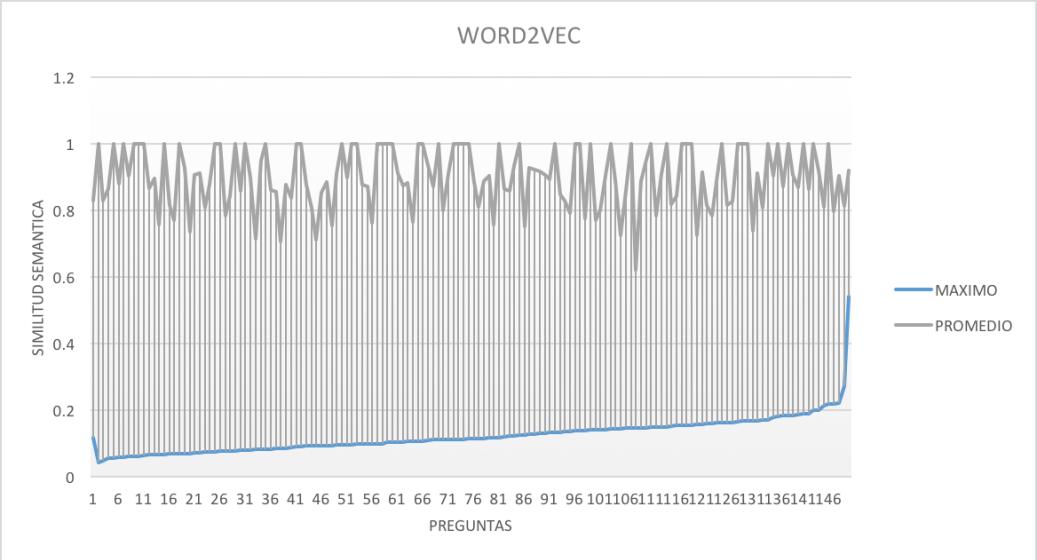

Gráfico 3.3.1.0: SST Word2Vec-SkipGram con la fórmula promedio y máximo.

GloVe

El algoritmo de Vectores Globales para representación de palabras, o más conocido como GloVe, es una extensión del método Word2Vec para el aprendizaje eficiente de vectores de palabras, desarrollo por Pennington, et al. en Stanford. GloVe es un algoritmo de aprendizaje no supervisado para obtener representaciones vectoriales de palabras. El entrenamiento se realiza en estadísticas globales de co-ocurrencia palabra- palabra agregadas de un corpus, y las representaciones resultantes muestran subestructuras lineales del espacio vectorial de palabras.
GloVe es un enfoque para unir las estadísticas globales de las técnicas de factorización matricial como Análisis Semántico Latente (ASL) con el aprendizaje basado en el contexto local en Word2Vec. En lugar de utilizar una ventana para definir el contexto local, GloVe construye una matriz xplícita de palabra-contexto o co-ocurrencia de palabras utilizando estadísticas en todo el corpus de texto. La metodología del modelo GloVe es crear primero una enorme matriz de co-ocurrencia palabra- contexto que consta de pares (palabra,contexto) de modo que cada elemento de esta matriz represente la

frecuencia con la que aparece una palabra con el contexto (que puede ser una secuencia de palabras). La idea es aplicar factorización matricial para aproximar esta matriz como se muestra en la figura 3.3.1.1.

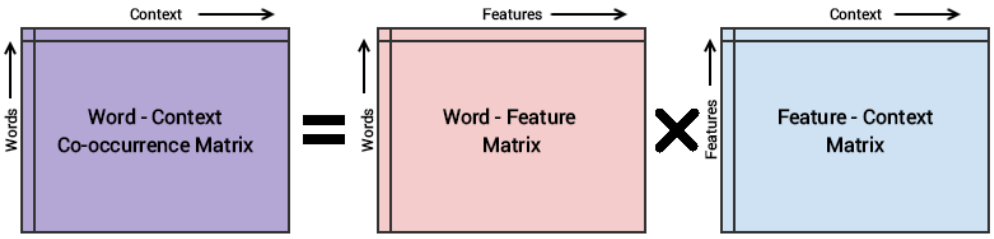

Figura 3.3.1.1: Modelo conceptual para la implementación del modelo GloVe.

La matriz **Word-Context (WC)**, la matriz **Word-Feature (WF)** y la matriz **Feature-Context (FC)**, se factoriza $WC = WF\ x\ FC$, el objetivo es reconstruir **WC** a partir de **WF** y **FC** multiplicándolos. Normalamente se inicializa **WF** y **FC** con algunos pesos aleatorios y se multiplica para obtener **WC'** (una aproximación de **WC**) y medir qué tan cerca está de **WC**. Se hace varias veces usando el Stochastic Gradient Descent (SGD) para minimizar el error. Finalmente, la **matriz Word-Feature (WF)** nos da las incrustaciones de palabras para cada palabra donde **F** se puede preestablecer en un número específico de dimensiones. Los modelos Word2Vec y GloVe son muy similares en su funcionamiento. Ambos tienen como objetivo construir un espacio vectorial donde la posición de cada palabra este influenciada por las palabras vecinas en función de su contexto y semántica. Word2Vec comienza con ejemplos individuales locales de pares de co-ocurrencia de palabras y GloVe comienza con estadísticas globales de co-ocurrencia agregadas en todas las palabras del corpus.

3.3.2. Algoritmo de GloVe

GloVe produce densas vectoriales words embeddings, donde las palabras que ocurren juntas están cerca en el espacio vectorial resultante. Si bien esto produce embeddings que son similares a Word2Vec (que tiene una gran implementación de Python en gensim), el método es diferente, en GloVe como

se menciona anteriormente produce embeddings factorizando el logaritmo de la matriz de co-ocurrencia de palabras de un dataset.

GloVe en Python

Las librerías que se requieren instalar para el funcionamiento de GloVe, son las siguientes:

- Scipy pip install scipy
- Numpy pip install numpy
- math pip install math3

Scipy es un software de código abierto para matemáticas, ciencias e ingeniería. La biblioteca SciPy depende de NumPy, que proporciona una manipulación de matrices N-dimensional rápida. La biblioteca SciPy está diseñada para trabajar con matrices NumPy y proporciona muchas rutinas numéricas eficientes y fáciles de usar, como rutinas para la integración y optimización numéricas.
El algoritmo GloVe consta de los siguientes pasos:

1. Dado un vocabulario, un corpus, un tamaño de ventana *(N)* y un recuento mínimo de co-ocurrencias, el algoritmo toma una ventana deslizante de un tamaño N y la pasa por todo el corpus, contando las co-ocurrencias de cada palabra con cualquier otra palabra. El resultado es la matrix dispersa X_{ij}.

2. Se inicializa los parámetros del modelo. Esas son la matriz de vectores de palabras W de tamaño *(V x D)* donde V denota el tamaño del vocabulario y D denota las dimensiones del vector especificadas; en un vector de sesgo para cada término. Cada término se inicia aleatoriamente en el rango de (-0.5, 0.5).

3. Las co-ocurrencias se bajaran y el algoritmo calcula la función de costo asociada con la fase inicial del algoritmo.

4. Para cada iteración, el gradiente de la función de costo J se derivan con respecto a los parámetros que se actualizan de acuerdo con una tasa de aprendizaje.

Algorithm 4: GloVe

```
   Input: datasetPreguntasFrecuentes, matriz-coocurrencia
   Output: SST de pregunta1 a preguntaN
 1 distancia-coseno-entre-dos-frases(p1, p2)
 2     devolver(1-scipy.spatial.distance.cosine(model[word1],model[word2]))
 3 preprocess(preguntas)
 4     remueve stopwords
 5 calcular-matrizHeat-para-dos-frases(p1,p2)
 6     p1 = preprocess(p1)
 7     p2 = preprocess(p2)
 8      resultado = [[distancia-coseno-entre-dos-palabras(word1,word2) for
     word2 in s2] for word1 in s1]
 9     devuelve resultado
10 for key in pregunta do
11 |    vector = pregunta[key] similitud =
   |     calcular-matrizHeat-para-dos-frases(vector[0],vector[1])
12 devuelve similitud
```

Resultados

En el gráfico 3.3.1.2 se muestran los resultados que se obtuvieron aplicando el algoritmo 4 con la fórmula del promedio y máximo. Como se puede visualizar ambos tuvieron el mismo resultado. Pero más adelante se compararán los resultados de cada medida de SST, para saber cuál de las tres medidas tuvieron un mayor resultado.

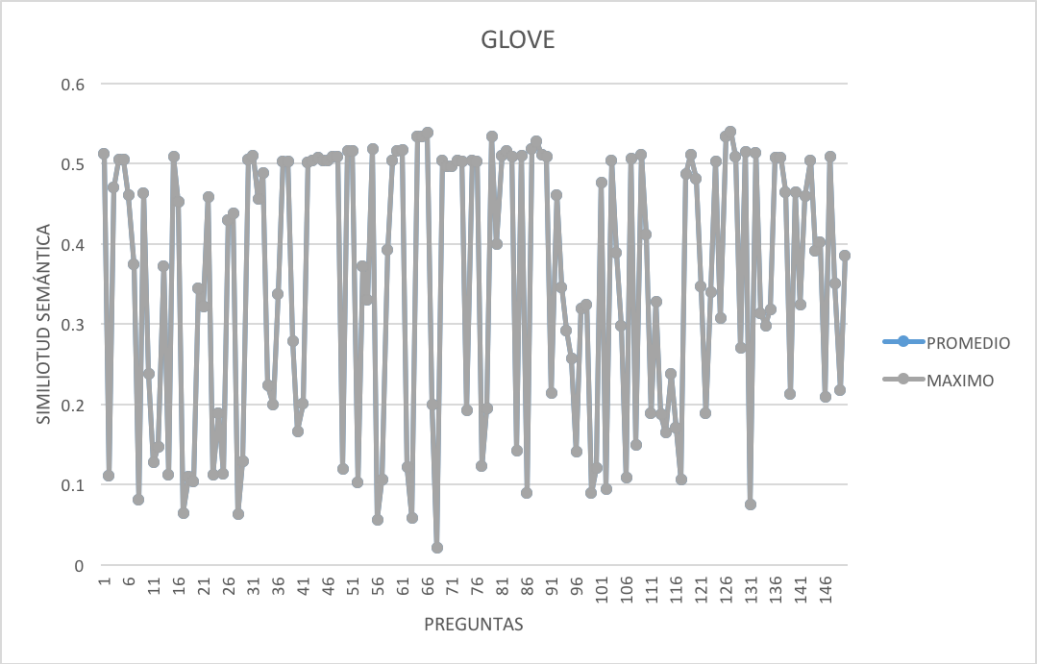

Gráfico 3.3.1.2: SST GloVe con la fórmula promedio y máximo.

FastText

El modelo FastText es una biblioteca del aprendizaje de word embeddings (representación de texto) y clasificador de texto, fue introducido por primera vez por Facebook en 2016. Su característica es la rapidez, que puede utilizarse como modelo de referencia para la clasificación de textos. En comparación con otros modelos de clasificación de texto, como regresión logística, red neuronal, FastText reduce en gran medida el tiempo de entrenamiento mientras mantiene el efecto de clasificación.

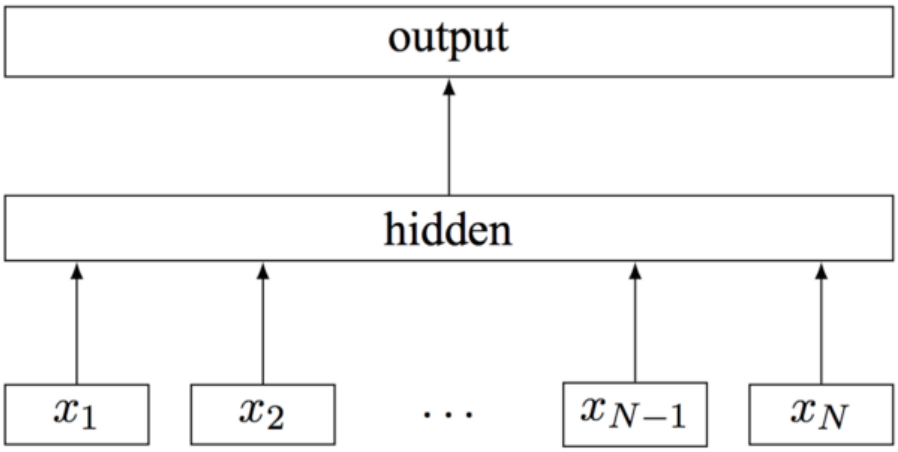

Figura 3.3.1.3: Modelo de arquitectura de texto rápido para una oración con N características de ngram $x_1, ..., xN$. Las características están integradas y promediadas para formar la variable oculta.

El modelo es una red neuronal simple con una sola capa. La representación de la bolsa de palabras del texto se introduce primero en una capa de búsqueda, donde se obtienen los embeddings para cada palabra. Luego, esos embeddings de palabras se promedian, para obtener una sólo embedding. En la capa oculta, se termina con un *n-words x dim* número de parámetros, donde *dim* está el tamaño de los embeddings y *n-words* el tamaño del vocabulario. Después del promedio, sólo tenemos un único vector que luego se alimenta a un clasificador lineal: se aplica el softmax sobre una transformación lineal es una matriz con *dim x n-output*, donde *n-output* es el número de clases de salida. La probabilidad logarítmica final es:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)), \tag{3.4}$$

donde:

- x_n es la representación original codificada de one-hot de una palabra (o característica n-gram),

- **A** es la matriz de búsqueda que recupera la palabra embedding,
- **B** es la transformación de salida lineal,
- f es la función softmax.

Representación de texto

Una principal idea del aprendizaje automático moderno es representar palabras mediante vectores. Estos vectores capturan información oculta sobre un idioma, como analogías de palabras o semántica. También se utiliza para mejorar el rendimiento de clasificadores de texto. FastText es una versión modificada de Word2Vec es decir, skip-gram y CBOW. La única diferencia entre FastText y Word2Vec es su combinación de estrategias (cuáles son la entrada, la salida y el diccionario del modelo).
En Word2Vec cada palabra se representa como una bolsa de palabras, pero en FastText cada palabra se representa como una bolsa de caracteres n-gram. Pero agregar un método de n-gram traerá al diccionario mucho vocabulario, lo que conducirá a un desbordamiento de la memoria. Para resolver estos problemas, FastText usa el método del hash buckets. El proceso de implementación específico se muestra en la figura 3.3.1.4. **Carácter n-grama** es la secuencia contigua de n elementos de una muestra dada de un carácter o palabra. Puede ser brigrama, trigrama, entre otros.

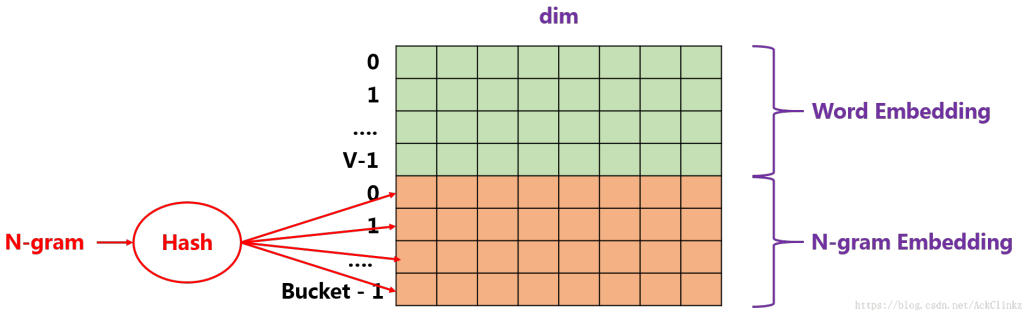

Figura 3.3.1.4: Proceso del Hash buckets.

Ejemplo de como convertir una entidad n-gram en una capa oculta. Un texto consta de tres palabras $\mathbf{w}_1 w1, \mathbf{w}_2 w2, \mathbf{w}_3 w3$. Su biagrama es $\mathbf{w}_1 2w12, \mathbf{w}_2 3w23$. Entonces, la fórmula de cálculo para la capa oculta es la siguiente:

$$hidden = \frac{1}{5}(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3 + \mathbf{w}_1 2 + \mathbf{w}_2 3) \tag{3.5}$$

Clasificador de texto

La clasificación de texto es un problema para algunas aplicaciones, como la detección de spam, el análisis de opiniones o las respuestas inteligentes. El objetivo de la clasificación de texto es asignar documentos (como correos electrónicos, publicaciones, mensajes de texto, reseñas de productos, etc.) a una o varias categorías. El enfoque dominante para construir tales clasificadores es el aprendizaje automático, es decir, aprender reglas de clasificación a partir de ejemplos. Para construir estos clasificadores, se necesita datos etiquetados, que consisten en documento y sus categorías correspondientes (tags or labels).

Algoritmo de FastText

Como ya se había mencionado anteriormente FastText se utiliza para clasificación de texto, puede entrenar grandes conjuntos de datos en minutos y encontrar similitudes semánticas. En este caso se implementará para poder obtener la similitud entre una oración y otra.

FastText con Gensim

Para uso de la biblioteca FasText se ocupa el módulo *gensim.models.fasttext*. Para la representación de palabras y similitud semántica, se usa el modelo Gensim para FastText.

Las librerías que se requieren instalar para el funcionamiento de FastText, son las siguientes:

- FastText pip install fasttext
- Keras pip install keras
- nltk pip install nltk
- tensorflow pip install tensorflow

El Algoritmo 5 lleva a cabo el proceso de obtener la similitud semántica entre una oración y otra. Recibe como entrada un conjunto de datos que contenga preguntas, cuyo objetivo será procesar la información para poder obtener como resultado la similitud. Crea la bolsa de palabras,para después hacer el modelo FastText con los datos ya procesados, donde *size* dimensionalidad de los vectores de palabras y el valor predeterminado es 100; *window* la distancia máxima entre una palabra de destino y palabras alrededor de la palabra de destino, la ventana predeterminada es 5; *min-count* es el recuento mínimo de palabras a considerar al entrenar el modelo, las palabras con ocurrencia menor que este recuento serán ignoradas, el valor predeterminado es 1; *sample* el umbral para configurar qué palabras de mayor frecuencia se muestran aleatoriamente, el rango útil es (0, 1e-5); *sg* el algoritmo de entrenamiento skip-gram si sg = 1, en caso contrario CBOW y por último *iter* número de iteraciones sobre el corpus.Con este modelo se obtiene la similitud semántica de cada palabra, para saber con exactitud la similitud por oración, se guardan los valores por cada palabra y al finalizar el recorrido de la oración se divide entre el total de palabras para así dar el resultado preciso de SST.

Algorithm 5: FastText

Input: datasetPreguntasFrecuentes
Output: SST de pregunta1 a preguntaN

```
word-tokenized-corpus = listOfWords(datasetPreguntasFrecuentes)
datosProcesados = procesar(datasetPreguntasFrecuentes)
modeloFastText = FastText(word-tokenized-corpus, size = 100, window
  = 5, min-count = 1, sample = 1e-2, sg = 1, iter=100)
for key in sentences do
    for s1 in preguntaUsr do
        for s2 in preguntaDoc do
            sumaSimilitud = modeloFastText.wv.similarity(s1.lower(),
             s2.lower()) count +=1
            guardaSumaSimilitud(sumaSimilitud)
        /* Devuelve la similitud de la pregunta del usuario y la pregunta
           del corpus                                                    */
        resultadoSimilitud = ( suma(guardaSumaSimilitud) / count)
        /* Se limpia la lista dónde se almacena la similitud de la palabra
           s1 y palabra s2                                               */
        clearguardaSumaSimilitud
```

Resultados

En el gráfico 3.3.1.5 se muestran los resultados obtenidos del algoritmo 5 aplicando las dos fórmulas. Se observa que aplicando la fórmula del máximo se obtiene mayor similitud que con el promedio. En el siguiente punto se muestra cuál medida de similitud semántica será la más óptima para aplicarlo en un chatbot de preguntas frecuentes.

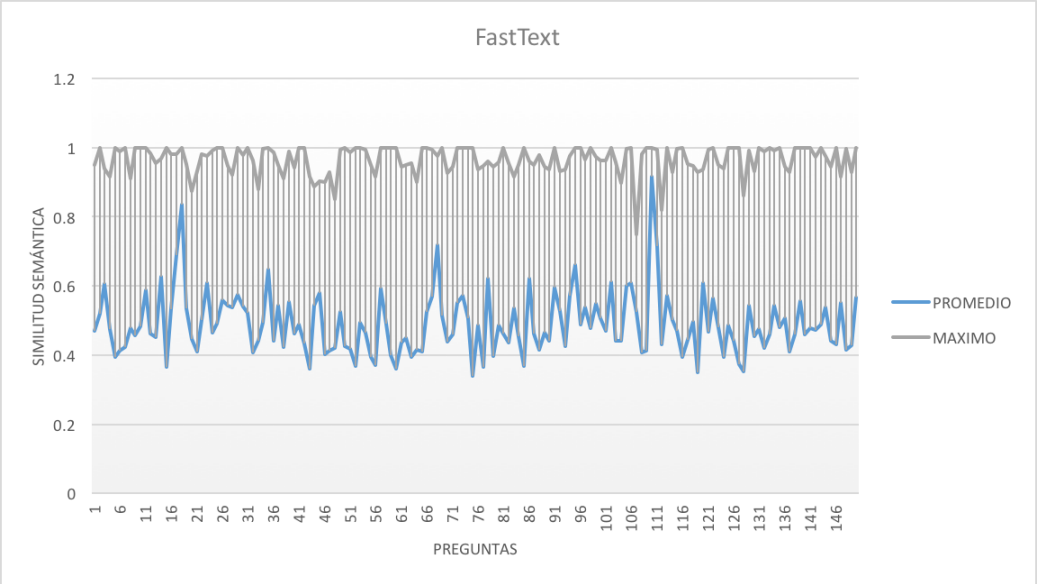

Gráfico 3.3.1.5: SST FastText con la fórmula promedio y máximo.

3.4. Evaluación de SST

En la fase anterior se evaluaron 3 medidas de SST para saber cuál de todos daba como resultado una mayor precisión y así aplicarlo a un chatbot de preguntas frecuentes. Los resultados obtenidos fueron aplicando la fórmula del máximo, ya que en los precedentes gráficos mostraban mayor similitud con dicha fórmula. En el gráfico 3.3.1.6 se unieron los resultados de las 3 medidas de similitud semántica (Word2vec,GloVe y FastTex) quien mayor resultado muestra es el modelo de FastText, con mayor rango de 0.74-1, a diferencia de Word2Vec su rango es de 0.62-1 y GloVe de 0.02-0.54.

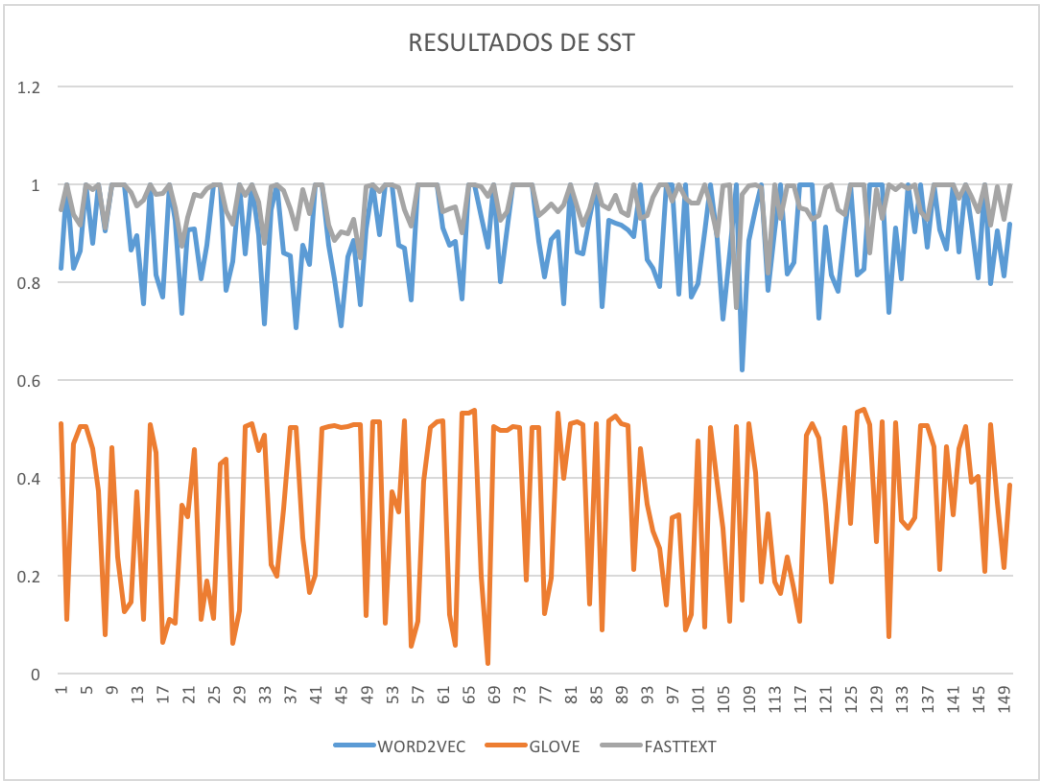

Gráfico 3.3.1.6: Resultados de Word2Vec, GloVe y FastText.

Como se muestra en el gráfico 3.3.1.7 la medida de SST que mejor resultado obtuvo en cuanto a la precisión (0.97) y en el gráfico 3.3.1.8 se muestran los 3 resultados obtenidos de los modelos anteriores, es **FastText**, debido a su estructura interna de las palabras mientras aprenden las representaciones de palabras, lo es que es muy útil para lenguajes morfológicamente y para palabras que ocurren muy esporádico.

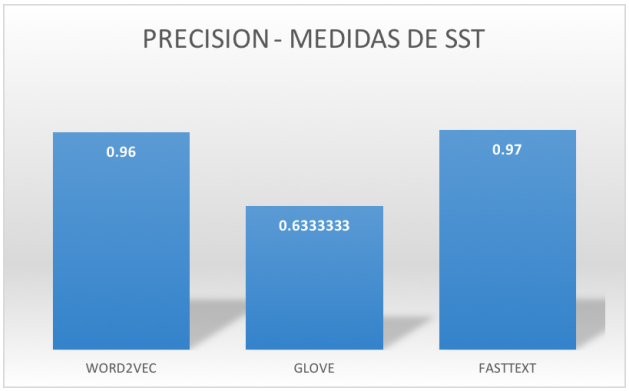

Gráfico 3.3.1.7: Precisión de Word2Vec, GloVe y FastText.

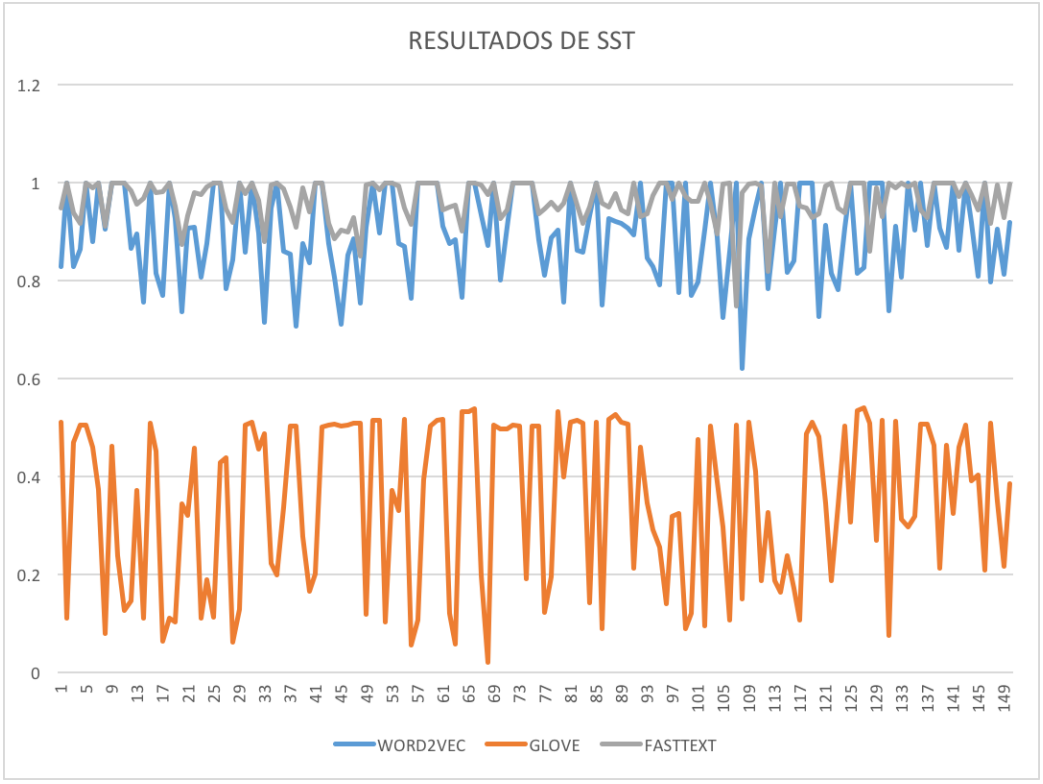

Gráfico 3.3.1.8: Resultados de modelos SST.

Con los resultados mostrados anteriormente se seleccionará el modelo de FastText para aplicarlo en un chatbot de preguntas frecuentes, ya que obtuvo mayor resultados en cuánto a la precisión de SST.

Ventajas de FastText de Word2Vec y GloVe

1. Es más rápido y sencillo de entrenar. En la evaluaci de similitud, FastText da mejores resultados que Word2Vec y GloVe en un conjunto de entrenamiento de 150 preguntas.

2. Se beneficia de la información de subpalabras. Si se da una palabra como "where", FastText usa todos los n-gramas de esta palabra para calcular puntuación.

3. Puede generar embeddings a patir de palabras fuera de vocabulario gracias a los n-gramas. Por lo que funciona bien con las palabras que son poco habitualese representan con precisión. GloVe es un algoritmo adicional para entrenar embeddings. Mientras que Word2Vec instenta capturar las coincidencias de palabras en una ventana a la vez.

4. Word2Vec como GloVe no proporcionan ninguna representación vectorial para palabras que no están en el diccionario modelo. Es una gran ventaja que tiene FastText.

5. A diferencia de FastText, el entrenamiento del modelo GloVe no utiliza una red neuronal, sino que calcula la matriz colineal mediante métodos estadísticos. Lo que hizo que en cuanto al tiempo se alentará un poco más.

Capítulo 4

Implementación y Resultados

4.1. Introducción

En el anterior capítulo se mostraron 3 medidas de SST con sus respectivos resultados, evaluando con un dataset de 150 preguntas, dando como un mejor resultado el modelo de FastText. Ahora en este capítulo, se presenta la última parte de la metología que se explicará más adelante, con el desarrollo de un chatbot de preguntas frecuentes desarrollado en python.

4.2. Metodología

Como se menciona en la sección 1.3, el objetivo general es implementar un algoritmo de SST aplicado en un Asistente Virtual. En base a lo anterior se propone la siguiente metodología (Figura 4.2.1):

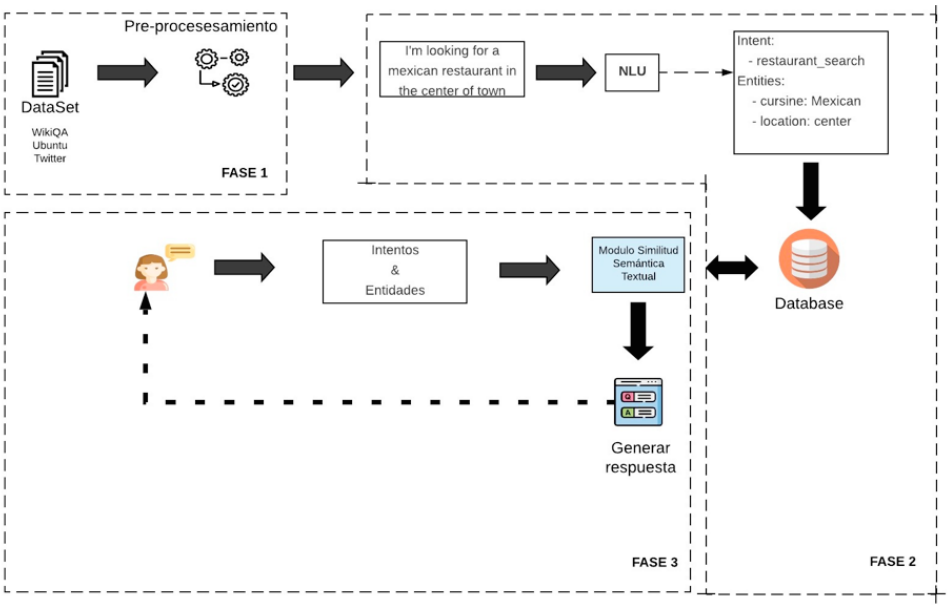

Figura 4.2.1: Metodología.

La metodología propuesta consiste en una medida de SST y su aplicación de un chatbot de preguntas frecuentes en el capítulo 3 se describieron las primeras dos, la fase 1 hace el procesamiento del dataset, eliminando aquellos símbolos especiales, entre otros. La siguiente fase consistía en analizar la información que proviene del dataset, identificando los intentos y entidades a nivel de sentencia. Pero en la revisión del estado del arte se encontraron algunos modelos de SST que se podían evaluar con el dataset y poder así obtener una diferencia entre cada uno, para posteriormente aplicarlo en un chatbot. Tan sólo trabajar con intentos y entidades, un chatbot ya podría regresar una respuesta, sin antes poder hacer un anális semántico para una mayor precisión de respuesta. Y por último la fase 3 es aplicar una medida de SST en un chatbot de preguntas frecuentes para poder una respuesta con mayor precisión a la necesidad del usuario.

4.3. Desarrollo del Chatbot de preguntas frecuentes

El chatbot se desarrollo en Python con el framework denominado Flask. Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo de número de líneas de código. Es basado en la especificación Web Server Gateway Interface (WSGI) de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD.
En la figura 4.2.2 se muestra el diagrama de secuencias general de la aplicación del chatbot. Al inicio de la aplicación el chatbot saluda y hace una pregunta *Hi, how can I help you?*, una vez que el usuario realiza un pregunta, el proceso de la app es obtener la pregunta del usuario, enviar la petición al servidor, este revisa si hay una petición, y si la hay compara si es una despedida, agradecimiento o saludo, si aplica cualquiera de las anteriores, responde el saludo y pregunta *¿Is there anything I can help you with?* como se muestra en la figura 4.2.3. Si no es un saludo, agradecimiento o despedida, compara el número de palabras de la pregunta y si es mayor a 1 manda a llamar al modelo de FastText, este modelo regresa la similitud y si es >=0.86 manda la respuesta como se visualiza en la figura 4.2.4; si no manda el siguiente mensaje *"Sorry, I can not help you with this question. Is there anything else I can help you with?"*. Finaliza la conversación cuando el usuario escribe *Bye o No thanks.*

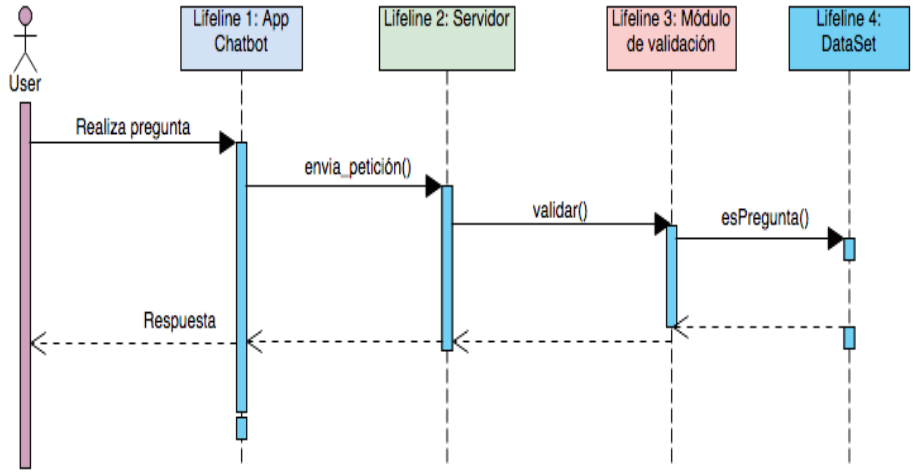

Figura 4.2.2: Diagrama de secuencias de la app chatbot.

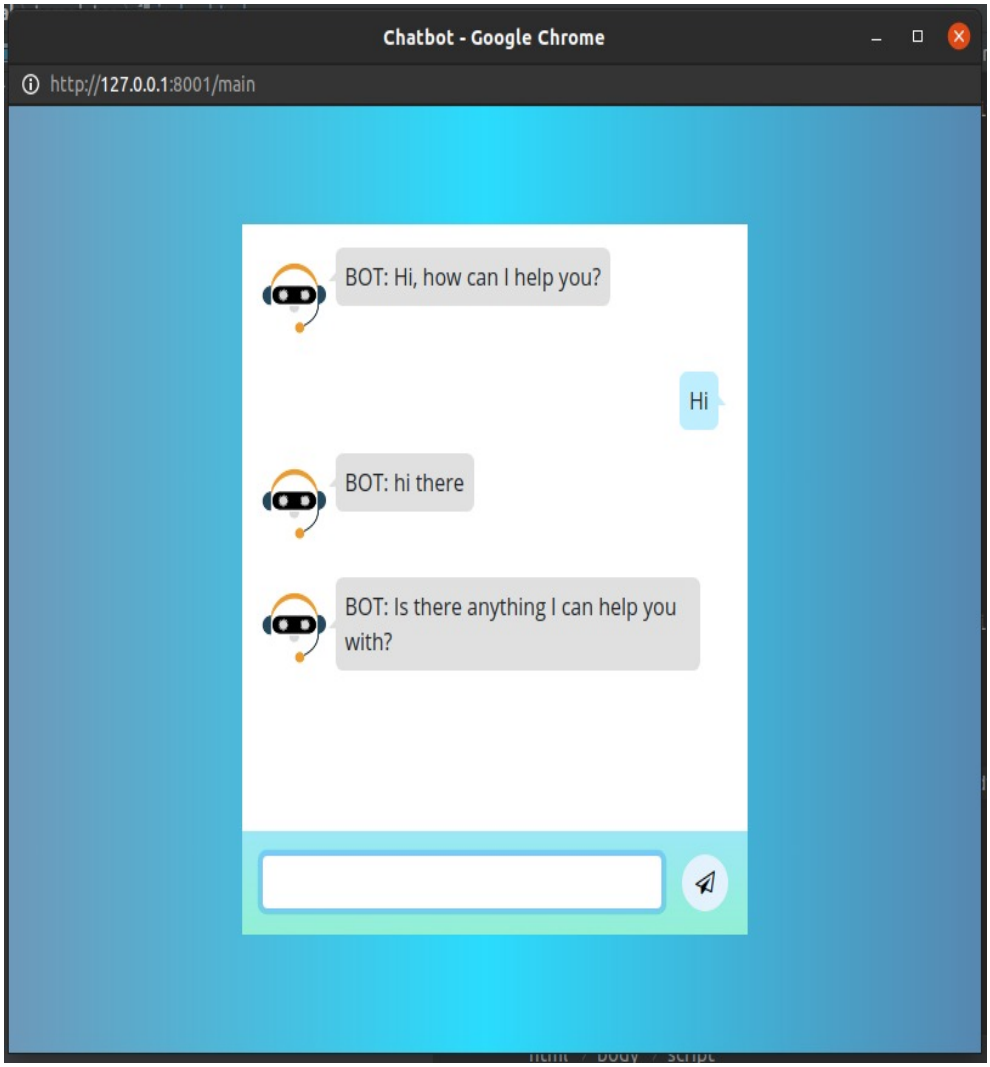

Figura 4.2.3: Chatbot saludando.

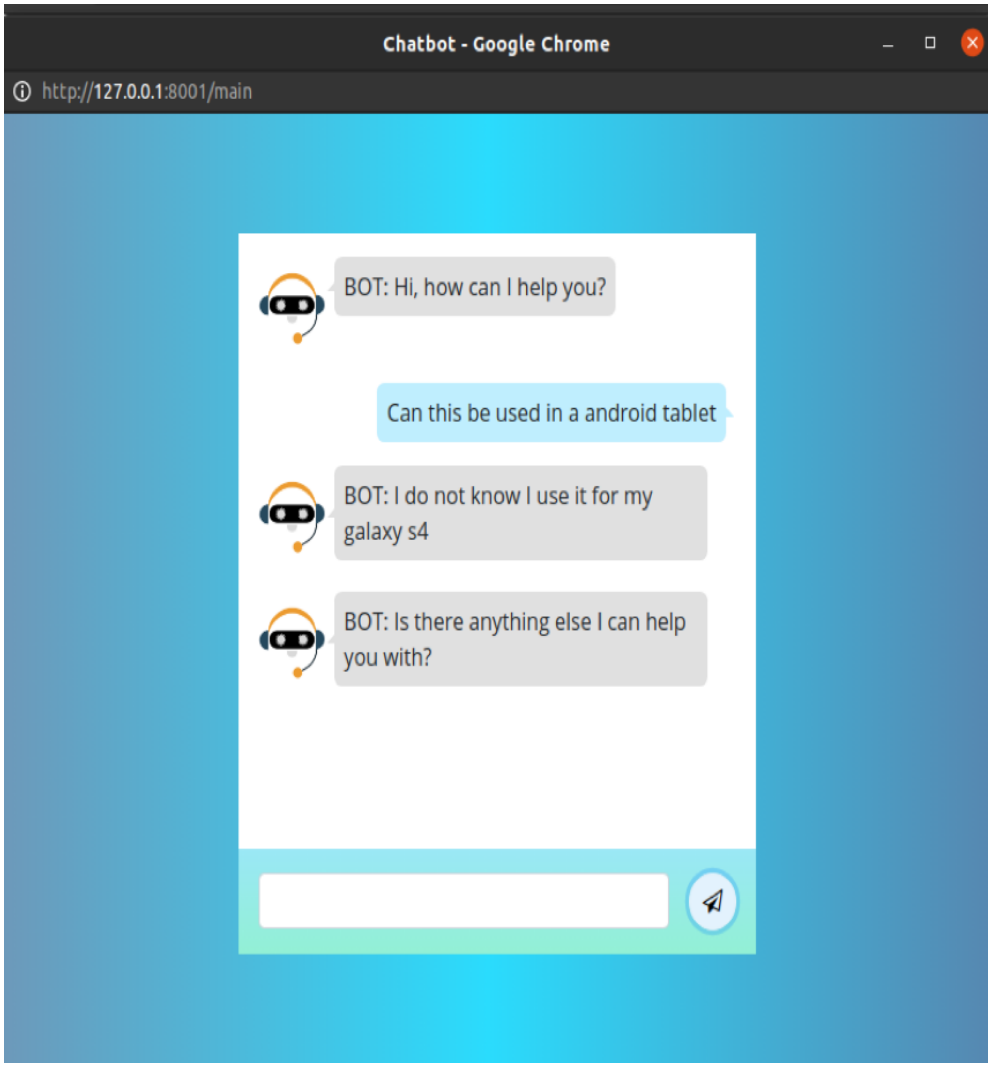

Figura 4.2.4: Chatbot respodiento la pregunta del usuario.

Requerimientos

Para la aplicación se requiere lo siguiente:

- Python 3.8
- Framework Flask 1.1

A continuación se muestra partes del código de la aplicación web.

1. Función principal para que la aplicación se ejecute como servidor web en la IP 127.0.0.1 con el puerto 8001.

```
if __name__ == '__main__':
    app.run(host='127.0.0.1', port='8001', debug=True)
```

Figura 4.2.5: Función principal.

2. send-message es la función donde se reciben las preguntas que escriba el usuario, recibe la pregunta en formato JSON, se extrae la pregunta del JSON y se le manda a la función chatbot-entry, donde se hace el procesamiento de la pregunta del usuario y envia la respuesta en formato JSON.

```
@app.route('/send', methods=['POST'])
def send_message():
    req = request.get_json()
    res = make_response(fq.chatbot_entry(req['message']), 200)
    return res
```

Figura 4.2.6: Función sent-message.

3. chatbot-entry recibe la pregunta del usuario, la cadena de caracteres se convierte a minúsculas y se guarda en la variable “user-response”, después se compara si el usuario escribió un saludo (hi,hello), despedida (bye, good bye) o agradecimiento (no thanks), si se cumple la condición responderá dependiendo si fue el saludo o cualquier de los otros ya mencionados. Si las condiciones anteriores no se cumplen se compara el número de palabras de la pregunta que sea mayor a 1, si se cumple

la condición la pregunta se enviará a la función donde se va a procesar la pregunta, en caso de no cumplir la condición responderá el chatbot "Can you write a question".

```
def chatbot_entry(string: str):
    user_response = string
    user_response = user_response.lower()
    usr_response = user_response.split()

    if user_response == 'bye' or user_response == 'no thanks' or user_response == 'see you later' or user_response == 'for now i say goodbye':
        return jsonify(message="Bye! take care..")
    else:
        if user_response == 'thanks' or user_response == 'thank you':
            return jsonify(message="You are welcome..")
        else:
            if greeting(user_response) != None:
                return jsonify(message=greeting(user_response), ask="Is there anything else I can help you with?")
            else:
                if len(usr_response) > 1:
                    return jsonify(message=semanticFastText(dicOracion, user_response), ask="Is there anything else I can help you with?")
                else:
                    return jsonify(message="Can you write a question")
```

Figura 4.2.7: Función chatbot-entry.

4. Una vez que haya pasado por el proceso de validación, se manda a llamar el modelo de FastText para la comparación de la pregunta del usuario con el dataset de la aplicación.

5. Por último se muestra la función que le envía la respuesta al usuario, si la petición es correcta.

```
fetch('/send', {
    method: 'POST',
    credentials: "include",
    body: JSON.stringify(data),
    cache: "no-cache",
    headers: new Headers({
        'Content-Type': 'application/json'
    })
}).then(res => res.json())
.then(res => {
    console.log(res);
    let myobje = document.getElementById("status");
    myobje.remove();
    bot_response = `<div class="row ml-2"><p class="t
    $(wrapper).append(bot_response);
    bot_ask = '<div class="row ml-2"><p class="text p
    if(res.ask)
    {
        $(wrapper).append(bot_ask);
        updateScroll()
    }
    updateScroll()
    index_msg++
})
.catch(error => console.error('Error:', error))
```

Figura 4.2.8: Función de envío.

4.4. Evaluación

La prueba de la propuesta del chatbot de preguntas frecuentes se evaluaron con 150 preguntas que se obtuvieron del dataSet para después en la aplicación de google llamada *paraphrase* pudiera parafrasear la pregunta orignal y de la pregunta parafraseada se simulara como una pregunta posible de un usuario, este procedimiento se llevo a cabo con cada pregunta original del dataSet

hasta llegar a las 150.
Para la evaluación del chatbot se utilizaron 3 métricas precisión, recall y accuracy, como se muestra en la gráfico 4.8.

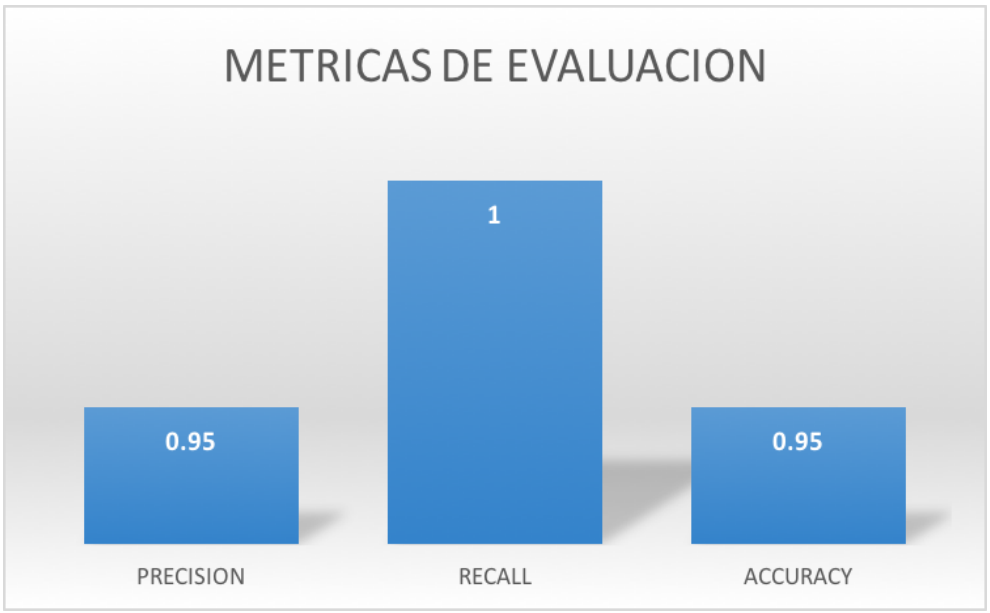

Figura 4.2.9: Gráfico de evaluación.

La propuesta del chatbot necesita una función de evaluación o métrica que no ayudará a estimar cuantitativamente la capacidad de generalización del modelo, es decir, su desempeño sobre la distribución completa de los posibles datos. Como se puede ver la app del chatbot tuvo una precisión de 0.95, recall de 1 y accuracy 0.95.

Capítulo 5

Conclusiones y trabajo a futuro

En este capítulo se presentan las conclusiones y el trabajo a futuro sobre SST aplicada a un AV. También se describe las limitaciones de la presente investigación.

5.1. Conclusiones

Debido a la situación actual del mundo y específicamente del país con la presente pandemia, esto produjo una alteración total desde la educación hasta las empresas. Pero gracias a la tecnología avanzada con programas que ayudan a la interacción entre usuario y computador, hacen que las empresas y escuelas no se detengan. Hoy en día un chatbot de preguntas frecuentes, puede ayudar mucho más a las empresas que ofrecen servicios o productos a interactuar más con sus clientes, brindándoles una mejor atención, asesorándolos o resolviendo sus dudas. Para que un chatbot pueda dar mayor servicio en cuanto a la precisión de sus respuestas, en este trabajo de tesis se propuso una metodología que consiste en 3 fases, la primera fue procesar un dataSet que contiene preguntas frecuentes de un apartado de electrónica en amazon; la segunda sobre la evaluación de medidas de SST; y la tercera poder aplicarlo en un chatbot de preguntas frecuentes para poder dar una respuesta más acertada. Se realizó una investigación de medidas de SST presentes en la literatura, de los cuales se evaluaron 3 Word2Vec, GloVe y FastText, quien obtuvo mayor precisión con 0.97 fue FastText. Con este resultado se pudo aplicar este modelo se aplico la tercera fase de la metodología con un chatbot de preguntas frecuentes, desarrollado en python, dando como resultado una

mejor respuesta al usuario.

5.2. Limitaciones

Estas son las limitaciones que presentaron en la investigación:

- Se evaluaron sólo 3 modelos de SST.
- Las medidas de SST sólo han sido evaluados con el idioma inglés.
- Las evaluaciones fueron con un dataset de 150 preguntas.

5.3. Trabajo a futuro

Las evaluaciones con los algoritmos de embeddings se hicieron con un conjunto de 150 preguntas con sus respectivas respuestas y se aplicaron a 3 algoritmos. Sin embargo, como trabajo futuro, se puede profundizar en la evaluación de otros algoritmos y combinarlo con el modelo de FastText. Y poder aplicarlo a un dataset que contenga más preguntas frecuentes. Otro avance sería aplicarlo en un sector ya sea educativo o empresarial.

Bibliografía

[1] Agirre, E., Banea, C., Cardie, C., Cer, D., Diab, M., Gonzalez, A.A., Guo, W., Mihalcea, R., Rigau, G. y Wiebe, J. (2014). SemEval-2014 Task 10: Multilingual Semantic Textual Similarity. ACM DL 81- 91.

[2] Aldarmaki, H. y Diab, M. (2016). GWU NLP at SemEval-2016 Shared Task 1: Matrix Factorization for Crosslingual STS. ACM DL, 663 – 667.

[3] Athreya, G., R., Ngonga, N.,A. y Usbeck, R. (2018). Enhancing Community Interactions with Data-Driven Chatbots–The DBpedia Chatbot. ACM DL, 1-4.

[4] Blanco, E. y Moldovan D. (2015). A Semantic Logic-Based Approach to Determine Textual Similarity. ACM DL, 23(4), 683- 693.

[5] Balodis, K., y Deksne, D. (Mayo,2019). "FastText-Based Intent Detection for Inflected Languages".

[6] Bowman, S., R., Angeli, G., Potts, C., Manning,C., D. (2015). A Large Annotated Corpus for Learning Natural Language Inference, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 632–642).

[7] Bojanowski, P., Grave, E., Joulin, A., Mikolov, T. (2016). Enriching Word Vectors with Subword Information, arXiv preprint. https://arxiv.org/abs/1607.04606.

[8] Cer, D., Diab, M., Agirre, E., Lopez, I. y Specia L., (2017).SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Cross-lingual Focused Evaluation. ACM DL, 1-14.

[9] Conneau, A., Kiela, D., Schwenk, H., Barrault,L., Bordes,A. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data, arXiv preprint. https://arxiv.org/abs/1705.02364.

[10] Foltz, P. W. (1996). "Latent Semantic Analysis for Text-Based". Behav. Res. Methods, Instruments Comput., vol. 28, no. 2, pp. 197–202.

[11] Gong Y., Zhao K., and Shu K. Q. (2016). "Representing Verbs as Argument Concepts". Proc. 30th Conf. Artif. Intell. (AAAI 2016), pp. 2615–2621, 2016.

[12] Hendrickx, S. N. Kim, Z. Kozareva, P. Nakov, S. Ó Séaghdha, Diarmuid, Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz (Jul. 2010). "Semeval- 2010 task 8: Multi-way classification of semantic relations between pairs of nominals", in Proc. 5th Int. Workshop Semantic Eval., Uppsala, Sweden, pp. 33–38.

[13] Jiepu, J. y James, A. (2017). Similarity-based Distant Supervision for Definition Retrieval. ACM DL, 1-10.

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

[15] Khurana, P., P. A., G. S., L. V., & A. S. (November de 2017). Hybrid BiLSTM-Siamese network for FAQ Assistance. ACM Digital Library, 9.

[16] Luceri, L., Deb, A., Badawy, A. y Ferrera, E. (2019). Red Bots Do It Better: Comparative Analysis of Social Bot Partisan Behavior. ACM DL, 1- 6.

[17] Mueller, J., and Thyagarajan, A. (2016). Siamese Recurrent Architectures for Learning Sentence Similarity. In AAAI Conference on Articial Intelligence (AAAI).

[18] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space, arXiv preprint. https://arxiv.org/abs/1301.3781.

[19] Pagliardini, M., Gupta, P., Jaggi, M. (2017). Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features, arXiv preprint. https://arxiv.org/abs/1703.02507.

[20] Pennington, J., Socher, R., Manning, C., D. (2014). Global Vectors for Word Representation, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532–1543).

[21] Rumshisky, A. (2008). "Resolving Polysemy in Verbs: Contextualized Distributional Approach to Argument Semantics". Distrib. Model. Lex. Linguist. Cogn. Sci. Spec. issue Ital. J. Linguist., pp. 1–27.

[22] S., B., Piper, A. M., & E. A. (June de 2019). "Hey Google, Do Unicorns Exist?": Conversational Agents as a Path to Answers to Children's Questions. ACM Digital Library, 13.

[23] Sandhaus,E. (2008). "The new york times annotated corpus", in Linguist. Data Consortium, Philadelphia, PA, USA.

[24] Sulem, D. H., Ari Rappoport, Z. A., & Omri Abend, L. C. (June de 2019). SemEval-2019 Task 1: Cross-lingual Semantic Parsing with UCCA. ACM Digital Library , 10.

[25] Supervised Learning of Universal Sentence Representations from Natural Language Inference Data (p. 5), por A. Conneau, D. Kiela, H. Schwenk, L. Barrault, A. Bordes., 2017, https://arxiv.org/abs/1705.02364.

[26] Semantic Sentence Similarity for Intent RecognitionTask (p. 5), por T. Brich, 2018.

[27] Semantic Sentence Similarity for Intent RecognitionTask (p. 12), por T. Brich, 2018.

[28] Torsten, Z., Levy, O., Gurevych, I., & Dagan, I. (2015). Ukp-biu: Similarity and entailment metrics for student response analysis. Atlanta, Georgia, USA, p 285.

[29] Wu, L., Fisch, A., Chopra, S., Adams, K.,Bordes, A., Weston, J. (2017). StarSpace: Embed All The Things!, arXiv preprint. https://arxiv.org/abs/1709.03856.

[30] Word2Vec Parameter Learning Explained (p.10), X. Rong., arXiv preprint, 2014, https://arxiv.org/abs/1411.2738.

[31] Yang, L., Chengjie, S., Lei Lin, Zhao, Y., y Wang, X. (November 2015). "Computing Semantic Text Similarity Using Rich Features".

[32] Zhao, Y., Nan, D., Junwei, B., Peng, C., Ming, Z., Zhounjun, L., y Jianshe, Z. (2016). DocChat: An Information Retrieval Approach for Chatbot Engines Using Unstructured Documents. ACM DL, 516- 525.