Trabajo de Fin de Grado

Grado en Ingeniería Informática

Análisis de sentimientos basado en opiniones turísticas

Sentiment analysis based in touristic reviews

Fernando Nantes-Machado Toledo

La Laguna, 3 de septiembre de 2018

Dña. **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora contratada doctora adscrita al Departamento de Ingeniería Informática de la Universidad de La Laguna, como tutora.

D. **José Luis González Ávila**, con N.I.F. 78.677.390-W profesor Asociado adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Análisis de sentimientos basado en opiniones turísticas"

ha sido realizada bajo su dirección por D. **Fernando Nantes-Machado Toledo**, con N.I.F. 79.085.621-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 3 de septiembre de 2018.

Agradecimientos

A mis padres y familia, que me han apoyado en algunos momentos difíciles durante el desarrollo de este trabajo.

A mis dos tutores, Isabel y José Luis, por todas esas reuniones y la ayuda que me han ofrecido durante el desarrollo de este trabajo, tanto cuando estaban en su horario como cuando no.

Licencia

Resumen

El objetivo de este trabajo ha sido la elaboración de un sistema que permite realizar el análisis de las opiniones elaboradas por los usuarios de diversos sitios como restaurantes, hoteles, parques y playas de toda la isla de Tenerife.

Las opiniones se obtienen a partir de una API que provee Google Maps que nos proporciona información valiosa del sitio como puede ser su localización, opiniones, idioma de éstas, etc. Todas estas opiniones serán almacenadas en una base de datos, de tal forma que más adelante sirvan como entrada al algoritmo de análisis. Los lugares elegidos fueron escogidos dentro de un radio que tiene como punto central un determinado lugar.

Se utilizan técnicas avanzadas de procesamiento de lenguaje natural para resolver el problema. Por una parte, la obtención de la representación de las palabras mediante vectores numéricos usando el algoritmo Word2Vec. Por otra, técnicas de aprendizaje automático orientadas a la clasificación del sentimiento de las reseñas con objeto de determinar si una opinión tiene una connotación positiva o negativa.

Palabras clave: Procesamiento de lenguaje natural, análisis de sentimientos, opiniones turísticas

Abstract

The aim of this project is the development of a system able to perform a sentiment analysis, taking as input many touristic reviews written by users who have visited places like restaurants, parks, beaches and hotels.

To get each one of the places, a radius was set having as its center one specific point. Thanks to this, information such as the location, name, etc. could be inserted into a database.

The reviews are obtained using a Google Maps API, which returns valuable information such as the place location, its reviews (with their language), etc. This data is later inserted into another database, which will be used to provide the input to the analysis algorithm.

Advanced techniques of natural language processing are used to solve the problem. On the one hand, getting the numeric vector representation of each review using Word2Vec algorithm. On the other hand, machine learning techniques used to classify the reviews whether they are positive or negative.

Keywords: Natural Language Processing, Sentiment Analysis, Touristic Reviews.

Índice general

Contenido

Índice de figuras

Índice de tablas

Capítulo 1 – Introducción

En la actualidad existen numerosos sitios web dedicados al turismo que ofrecen recomendaciones, información de lugares, opiniones, etc. Todas estas características influyen en las decisiones que toman los turistas a la hora de decidir su itinerario en un viaje.

El 75% de los viajeros reserva sus vacaciones a través de canales online, independientemente de sus conocimientos digitales (Álvaro Gutiérrez, 22 julio 2015). Sin embargo, todavía existe un 26% de la población que consulta las agencias tradicionales, en la mayoría de los casos, para complementar su búsqueda en internet. El reto de las agencias de viajes es apostar por la innovación y la multicanalidad para aprovechar ese tráfico a las agencias físicas.

A través de ambos canales, los viajeros encuentran destinos, consejos, ofertas y la posibilidad de planificar de forma integral sus vacaciones. Pero los principales motivos por los que el consumidor acude al canal online para reservar sus vacaciones son el precio y la facilidad en la reserva. Para quienes buscan diversión y entretenimiento, descubrir nuevos destinos o ir de aventura, las redes sociales son la fuente de información más recurrente para inspirarse o buscar opiniones de otros usuarios.

Estas fuentes de información se están convirtiendo cada vez más en recursos fundamentales para los departamentos de marketing en las empresas del sector (Big Data. Retos y oportunidades para el tursimo, s.f). Destaca su explotación mediante técnicas de procesamiento de lenguaje natural y aprendizaje automático orientadas a la segmentación de campañas, la fidelización de clientes, identificación de tendencias, etc. Este campo ha experimentado un gran auge gracias a la facilidad de acceso a las tecnologías Big Data, presentes en la mayor parte de análisis inteligente de datos en la actualidad.

Objetivos

El objetivo principal de este proyecto no es otro que la recopilación de información de lugares turísticos de la isla de Tenerife. En concreto, los datos que interesan son algunos como localización, nombre, tipo de sitio (restaurante, playa, etc.) y lo más importante: las opiniones en español de los usuarios, las cuales serán almacenadas en una base de datos para su posterior análisis.

Con la intención de lograr este objetivo principal, se ha tenido que llevar a cabo un estudio y trabajo previo sobre las herramientas, técnicas, algoritmos, etc. que se han utilizado debido a que algunas eran casi desconocidas. Este estudio ha permitido que se pueda realizar correctamente el procesamiento de lenguaje natural sobre las opiniones, aunque en algunos casos ha presentado dificultades, debido mayoritariamente al lenguaje coloquial que los usuarios utilizan a la hora de escribir y publicar dichas opiniones.

Integrando todas las herramientas y conocimientos necesarios, se desarrolla una plataforma capaz de obtener información sobre los sitios y las opiniones de cada uno de ellos, realizar un análisis morfológico y una conversión a forma canónica, y posteriormente el análisis de sentimientos que devolverá como resultado la valoración positiva, negativa o neutra de la opinión.

Alcance

Para lograr el objetivo del proyecto se realizan una serie de estudios acerca de la tecnología idónea para alcanzarlo. Al encontrar las óptimas el desarrollo, se procederá a ponerlas en práctica. Esto significa crear un sistema que sea capaz de captar las opiniones de determinados sitios cada día y añadir aquellas que hayan sido escritas recientemente, de tal forma que la base de datos en la que se almacenarán no deje de crecer, y hacerlo lo más parecido a un sistema Big Data posible. Con estas opiniones, se realizará un análisis de sentimientos para extraer qué lugares han sido de agrado para los clientes y cuáles no. A continuación, se exponen más específicamente cada una de las tareas a realizar:

- Estudio y análisis de las tecnologías a utilizar.
- Determinar las fuentes de las que se obtendrán las opiniones de los usuarios de distintos lugares de interés turístico (playas, parques, restaurantes, etc.)

- Estudio y análisis de las tecnologías de procesamiento de lenguaje natural a utilizar.
- Obtención y preprocesamiento de los datos.
- Implementación de las técnicas de análisis de sentimientos a aplicar.
- Evaluación de resultados, mediante la comparación con las puntuaciones otorgadas por los usuarios de Google Maps.

Antecedentes

Como antecedente directo de este trabajo tenemos el Trabajo de Fin de Grado del Grado en Ingeniería Informática: "Procesamiento de Lenguaje Natural y su aplicación en servicios de hostelería" realizado por José Gregorio Mesa Reyes en el curso 2015-2016. El objetivo del trabajo era la elaboración de un sistema capaz de interpretar el lenguaje natural según la información extraída de una web de hoteles, para posteriormente llevar a cabo una clasificación de servicios ofrecidos y de satisfacción de cada uno de ellos por parte de los huéspedes (José Gregorio Mesa Reyes, 15 febrero 2017).

El proyecto usó tecnologías de webscrapping como "UrlLib", y para el procesamiento de lenguaje natural se usaron algunas como "Spaguetti Tagger" y "TreeTagger Wrapper". En el trabajo se desarrolló el backend que contenía la parte del servidor, así como la base de datos georreferenciada; por otro lado, también se incorporó un frontend desarrollado en Angular.

Por otra parte, existen sitios web dedicados a los análisis de sentimientos (como http://www.tweetfeel.com que nos realiza un análisis basado en los Tweets de los usuarios sobre un determinado tema) que trabajan con opiniones obtenidas a través de otros sitios dedicados al turismo, en donde los usuarios pueden aportar su valoración y experiencia.

Otro ejemplo es el de http://hotelyou.es, una web que se dedica a procesar la descripción que un usuario especifique sobre el hotel que busca, y el sitio es capaz de analizarlo y mostrar resultados en base a las preferencias de la persona.

Destinatarios

Un perfil claro de destinatarios consiste en los clientes y usuarios potenciales de sitios y servicios turísticos, los cuales pueden usar la herramienta como punto de información y así observar resúmenes acerca de las opiniones que otras personas han aportado sobre el destino.

Una parte fundamental de este proyecto es la investigación de técnicas y herramientas para poder desarrollar sistemas de análisis del lenguaje aplicado al ámbito turístico. Debido a que se ha creado esta plataforma desde cero, y no se ha partida de una ya creada para poder mejorarla o experimentar con ella, todo el grueso del trabajo conlleva la exploración de las diferentes opciones y herramientas disponibles, así como la fase de aprendizaje en la que se ha aprendido utilizarlas. Es por esto que también este trabajo de fin de grado está destinado a aquellos desarrolladores que necesiten información sobre la implementación de un sistema de análisis de sentimiento con tecnología actualizada.

En resumen, esta herramienta resulta útil a aquellos usuarios que quieran conocer la satisfacción de otras personas con respecto a un determinado lugar de interés turístico, así como a quienes quieran implementarla.

Capítulo 2 – Estudio previo

Constituye la parte más extensa del desarrollo del proyecto. El estudio que se ha realizado de manera previa ha cubierto la mayor parte del tiempo y del trabajo realizado.

Para poder hacer frente a los objetivos propuestos, se ha realizado un análisis acerca de muchos puntos, tanto a nivel de lenguaje de programación como en el ámbito del tratamiento del lenguaje natural; conceptos, metodologías de trabajo, posibles herramientas y módulos a utilizar, capacidades y filtros de las distintas librerías y salidas correspondientes, etc.

Por último, dado que el idioma de las opiniones que se procesaron es únicamente el español, se invirtió tiempo en aprender a configurar correctamente la herramienta de análisis para que detectase el idioma y se realizase el procesamiento adecuadamente.

Definición del problema

El problema que se aborda en este proyecto consiste en la obtención de las opiniones de los clientes de diferentes lugares de interés turístico de la isla de Tenerife. A dichas opiniones se les aplicarán técnicas de aprendizaje automático o machine learning para obtener unos resultados con los que compararlos con las valoraciones aportadas por los usuarios en Google Maps.

Las opiniones que se van a analizar serán similares a:

“Buen restaurante, ambiente acogedor y tanto el servicio como el trato fueron excelentes.”

Sin embargo, dado que los textos que se usarán han sido escritos en su totalidad por usuarios, esto plantea el primer problema: la ortografía. Las herramientas de procesamiento de lenguaje natural que se usarán no están preparadas para una escritura coloquial y poco correcta, de tal forma que en algunas opiniones será obligado suprimir algunas palabras de tal forma que el análisis al menos no dé

problemas.

Análisis de los textos tratados

Los textos que se usarán, como ya se ha comentado, serán las opiniones que los usuarios agreguen a determinados sitios de interés turístico (en este caso, restaurantes, playas, hoteles y parques). Una vez se hayan obtenido dichas opiniones y almacenado en una base de datos, se usará una herramienta que convertirá cada palabra de la opinión a su forma canónica. Por ejemplo, la opinión que se ha usado en el apartado anterior quedaría de la siguiente manera después de la transformación:

"Buen restaurante, ambiente acogedor y tanto el servicio como el trato ser excelente."

Como vemos, convertir la oración a forma canónica no es más que obtener la forma "neutra" de cada una de las palabras que la conformen, sea adjetivo, sustantivo, adverbio, verbo, etc.

Una vez obtenida la versión compuesta exclusivamente por formas canónicas de cada una de las opiniones, se realizará entonces una conversión al formato vectorial. Es decir, cada una de las oraciones se verá representada como un vector de dimensión "n" formado por números. Estos vectores se obtienen a partir del entrenamiento previo de un modelo del lenguaje, en el que palabras similares según alguna característica lingüística (por ejemplo, semántica) estarán cercanas en el espacio vectorial real generado a partir del entrenamiento de una red neuronal. La opinión será representada por el vector promedio de todas las formas canónicas que la componen (Giatsoglou, M. V., 2017). Esta conversión nos interesa sobre todo para el algoritmo de aprendizaje automático que se aplicará, y con el cual podremos obtener una valoración numérica que se comparará con la establecida por los usuarios.

Procesamiento de lenguaje natural

Se ha requerido de un estudio extenso acerca de estas técnicas. Comprender bien los conceptos y cómo se desgrana la información obtenida una vez se analiza cualquier tipo de texto es básico para poder trabajar sobre ello.

Existen diversos conceptos implicados en el procesamiento del lenguaje que es necesario entender y que se usarán a lo largo de este documento.

- **Lexicón:** serie abstracta no ordenada de palabras pertenecientes a un lenguaje, una persona o una región, así como las reglas que permiten combinar las mismas.
- **Etiquetado gramatical (Part-Of-Speech Tagging):** también llamado POS Tagging, es un etiquetado de las palabras de un texto según su categoría gramatical. Puede presentarse de dos maneras diferentes, según se realice en base a la definición propia de la palabra o bien en base a la función que desempeña en su contexto. Por ejemplo:

  "El guapo entró en la sala."

  Según la definición propia de la palabra "guapo", está claro que es un adjetivo masculino, sin embargo, atendiendo al desempeño que realiza en la oración, se ve que es un sustantivo.

  Dado que no existe un convenio establecido ni una única manera de etiquetado, el que se haya clasificado bien la palabra o no atendiendo a su contexto depende la herramienta utilizada, y por tanto cada una ofrecerá diferentes resultados.

  A modo de ejemplo, se presenta una Figura 1 que corresponde con la tabla de las etiquetas usadas por la librería de procesamiento de lenguaje natural, Freeling, para cada tipo de palabra del español.

Part of Speech: `adjective`

Position	Atribute	Values
0	category	**A**:*adjective*
1	type	**O**:*ordinal*; **Q**:*qualificative*; **P**:*possessive*
2	degree	**S**:*superlative*; **V**:*evaluative*
3	gen	**F**:*feminine*; **M**:*masculine*; **C**:*common*
4	num	**S**:*singular*; **P**:*plural*; **N**:*invariable*
5	possessorpers	**1**:*1*; **2**:*2*; **3**:*3*
6	possessornum	**S**:*singular*; **P**:*plural*; **N**:*invariable*

Part of Speech: `conjunction`

Position	Atribute	Values
0	category	**C**:*conjunction*
1	type	**C**:*coordinating*; **S**:*subordinating*

Figura 1

- **Reducción de palabras:** sea cual sea el lenguaje analizado, es muy común el uso de palabras derivadas, algo que se debe tener en cuenta para evitar duplicar palabras iguales (por ejemplo, *bonito* y *bonitos*), que desde el punto de vista semántico deben considerarse idénticas. Existen dos técnicas para la reducción de palabras:
    - **Lematización:** consiste en hallar el lema correspondiente a dicha palabra (también llamado como la entrada estándar de dicha palabra en un diccionario). Así, se puede sintetizar las palabras similares, ya sea por medio de una lematización puramente morfológica o, en caso de ubicarla según su contexto, a través de un análisis sintáctico. Es la técnica que se usará con cada palabra de las opiniones en este proyecto.
    - **Stemming:** se basa en reducir una palabra a su raíz. Es muy utilizado en implementaciones de búsquedas de bases de datos y de navegadores, a fin de aumentar los resultados potenciales.
- **Análisis sintáctico (parsing):** aunque esta técnica no haya sido usada en este proyecto, es conveniente definir sus funciones, las cuales son llevar a

cabo un análisis sintáctico, obteniendo como resultado un árbol donde aparece cada palabra en el nivel correspondiente junto con la etiqueta que representa su función en la oración.

Word embedding

Son un conjunto de técnicas de procesamiento de lenguaje natural en donde las palabras o frases del vocabulario son vinculadas a vectores de números reales de la forma:

[1.2342, 2.2342, 0.2343, -1.2342, 5.2342, 4.2324, …]

Esto implica que palabras con significado similar o función sintáctica similar tengan una representación vectorial cercana en el espacio numérico. Para lograrlo, es posible usar varios métodos entre los cuales destacan las redes neuronales, modelos probabilísticos, etc. Estos métodos se rigen por el principio: "*si dos palabras tienen contextos muy similares, resultará entonces que esas dos palabras tendrán significados parecidos o que al menos estén relacionadas*" (Word Embedding, s.f). Es por esto que se dice que se basan en el contexto de la palabra.

Word2Vec

Word2Vec se define como un grupo de modelos relacionados usados para obtener "word embeddings". Recibe como entrada un gran corpus de texto con el cual produce un espacio vectorial, generalmente de una dimensión mayor de 100 en la que a cada palabra del corpus que cumpla unas características determinadas se le asigna su vector correspondiente en el espacio. A las palabras se les asigna la representación en el espacio vectorial de tal manera que aquellas que compartan contextos similares en el corpus, estarán próximas entre sí en dicho espacio.

Este algoritmo trabaja mediante una red neuronal donde una de sus capas está oculta. Además, puede utilizar dos arquitecturas de modelo: Continuous Bag Of Words (CBOW) y continuous skip-gram. En la primera arquitectura, el modelo predice la palabra a partir de una ventana de palabras usadas como contexto. Por otro lado, en la segunda arquitectura el modelo usa la palabra para predecir el contexto (Word2Vec, s.f) (Mikolov, 2013). La red se entrena para obtener el vector de características, la predicción de palabras o contextos según el caso, es un objetivo secundario.

Por último, mencionar que es una simplificación del modelo propuesto por Yoshua Bengio para que resulte ser más eficiente. (Yoshua Bengio, 2003)

Análisis de sentimientos

Conocido como “sentiment analysis”, se refiere al uso del procesamiento del lenguaje natural para identificar y extraer información subjetiva de los textos. Se trata de una tarea de clasificación masiva de documentos de manera automática, en función de la connotación positiva o negativa del lenguaje utilizado en el documento.

La forma más habitual de realizar un análisis de este tipo es mediante la denominada polaridad, a través de la cual se clasifica el mensaje en función de la intención que tenga el autor al escribirlo, pudiendo ser positivo, neutro o negativo. La manera sencilla de hallar esta polaridad es mediante la comparación. Se establecen tres conjuntos de palabras o frases: positivas, neutras y negativas. Esto se utiliza para que cualquier texto que contenga esas palabras o combinaciones de ellas, quede automáticamente encuadrado en una categoría de una forma previamente definida o descartado directamente. Por ejemplo, mensajes que contengan “No me gusta”, “desagradable” o “no se recomienda” quedarán marcados como negativos, y aquellos con “Buen ambiente”, “cómodo” o “perfecto” quedarán marcados como positivos. (Análisis de sentimiento, ¿qué es, cómo funciona y para qué sirve? 19 julio, 2017)

Otra posibilidad consiste en realizarlo mediante el uso de word embeddings. En este caso, cada palabra del mensaje escrito por el usuario es convertida a un vector de dimensión “n” y añadido a una lista. Cuando el mensaje no contenga más palabras, la lista de vectores resultantes será una matriz, a la cual se le aplicará la media por columnas. De esta forma, el resultado será un vector de dimensión “n” que representará al mensaje completo.

Support Vector Machines (SVMs)

Son un conjunto de algoritmos de aprendizaje supervisado aplicado a problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento, es posible entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. En el caso de este proyecto, se proporcionará como conjunto de entrenamiento opiniones en formato de word embedding junto a las valoraciones que lo usuarios escribieron. Tras esto, se aplicará sobre el conjunto de prueba formado por el resto de las opiniones codificadas en forma vectorial para que el algoritmo genere las valoraciones a partir de lo que ha aprendido en su entrenamiento previo. En este conjunto se comparan las obtenidas con las esperadas (las que los usuarios otorgaron).

La base teórica matemática para el caso más simple de tener 2 clases es la

siguiente: una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases por 2 espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector entre los 2 puntos, de las 2 clases más cercanos al que se llama vector soporte. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, pueden ser clasificadas asignadas a una o la otra clase. (Máquinas de vector de soporte, s.f)

Validación cruzada

La metodología de la división de conjuntos utilizada en SVMs posee una debilidad a la hora de elegir la proporción de cada conjunto con respecto al original. La evaluación y obtención de resultados puede ser diferente dependiendo de cómo sea la división entre datos de entrenamiento y prueba. Debido a estas carencias aparece el concepto de validación cruzada.

A pesar de que existen diversos métodos de validación cruzada, en este proyecto se ha decidido utilizar el método de validación cruzada de K iteraciones. Este consiste en, utilizando los datos al completo (es decir, sin divisiones en conjuntos de prueba y entrenamiento hechas por el usuario), crear K subconjuntos (donde el valor de K sí es especificado por el usuario, y depende de la medida del conjunto de datos). Uno de los subconjuntos es usado como datos de prueba y los K-1 restantes como datos de entrenamiento. El proceso de validación cruzada es repetido durante K iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Al finalizar, se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado. Este método es muy preciso pues evaluamos a partir de K combinaciones de datos de entrenamiento y de prueba, aunque cuenta con la desventaja de que es muy lento computacionalmente. (Cross-validation: evaluating estimator performance, s.f)

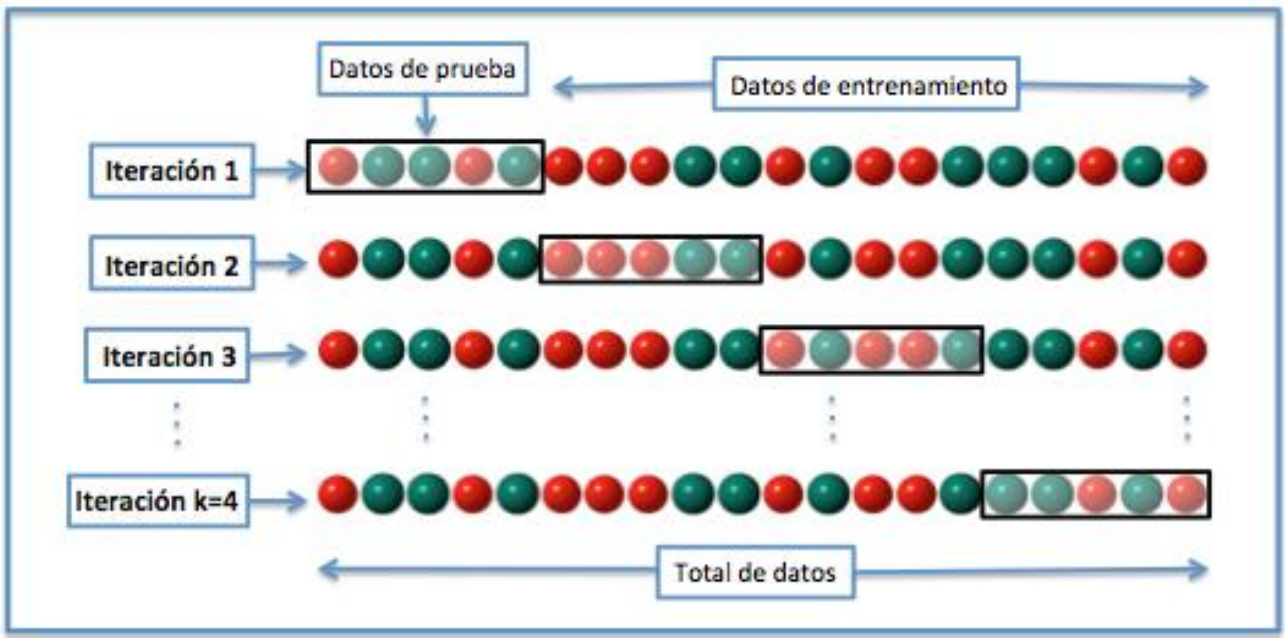


Figura 2

Tecnologías usadas

Python

Inicialmente se dio libertad a la hora de elegir lenguaje de programación para realizar el proyecto. Sin embargo, hay determinados lenguajes que están mejor preparados para el procesamiento de lenguaje natural que otros (entre ellos, Java, R, Python, etc.) Dado que aparte de querer aprender con el trabajo desarrollado en este proyecto, también es interesante aprender diversos lenguajes, la elección fue Python, puesto que es un lenguaje bastante usado, con una documentación muy amplia y comunidad muy activa.

Python es un lenguaje de programación interpretado, de alto nivel y multipropósito. Su sintaxis es fácil de entender si ya se posee un poco de experiencia con otros lenguajes como por ejemplo Ruby o Javascript. Soporta varios paradigmas de programación como orientación a objetos, programación funcional e imperativa, es multiplataforma y utiliza tipado dinámico (también llamado lenguaje débilmente tipado). (Python, s.f)

Sin embargo, también se requirió de un tiempo de aprendizaje sobre este lenguaje, dado que aunque pueda parecer intuitivo, esconde algunas diferencias con respecto a los otros lenguajes como: el tipado dinámico, la utilización de módulos o la forma de recorrer listas. El lenguaje se utilizó en un entorno Ubuntu Linux 16.04, utilizando la versión 3 del intérprete de Python. Esto es debido a que dicha versión

permitía una mejor visualización de las cadenas obtenidas de la API de Google Maps (puesto que la versión 2.x de Python añade caracteres "\u" para indicar que una cadena de texto está en formato Unicode). Como IDE se ha usado Microsoft Visual Studio Code.

pip

"pip" (en concreto su versión para Python3, "pip3") es un sistema de gestión de paquetes utilizado en Python que sirve para poder instalar y hacer uso de las distintas librerías y módulos que existen para este lenguaje. (pip, s.f) Gracias a este sistema, se han instalado librerías como pymongo, requests, gensim, etc. de las cuales se hablará más adelante.

Google Maps API

Google Maps posee una API a disposición de los programadores capaz de ofrecer información sobre los lugares que se deseen. Su modo de uso consiste en lo siguiente:

- Es necesario poseer una clave de API, única para cada usuario y es Google quien la asigna. Es de uso personal y gracias a ella se puede tener un control sobre las peticiones realizadas. (Get API Key, s.f)
- También hace falta lo que se llama "place_id", lo cual es un identificador alfanumérico único de cada lugar gracias al cual Google es capaz de conocer el lugar que se le pide y devolver la información asociada.
- Para automatizar la obtención de los place_id y no tener que obtenerlos uno por uno, Google facilita una URL en la que se le especifica las coordenadas de un lugar, un radio en kilómetros, la clave de API y una palabra clave (como "hotel", "restaurante", etc.). A partir de estos datos, Google Maps devuelve información (incluido el place_id) en formato JSON de todos los lugares que concuerden con la palabra clave indicada, y que estén dentro del radio establecido que tenga como centro las coordenadas indicadas.
- Con estos datos, la API devuelve un JSON que contiene el nombre del lugar, dirección, horarios, coordenadas, fotos, opiniones, etc. (Place details, s.f)

Requests

Requests es una librería para Python que permite realizar peticiones http desde el código y obtener en un objeto “res” la respuesta que devuelva el servidor. Posee una función llamada “get()” a la que se le pasa la URL a la cual queremos hacer la petición, y la respuesta se almacena en un objeto “res”. (Requests: HTTP for Humans, s.f)

MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, desarrollada bajo el concepto de código abierto. El término “NoSQL” significa que los datos no se guardan de forma estructurada en una tabla como hacen las bases de datos típicas “SQL”, sino que Mongo guarda estructuras de datos en documentos similares a JSON con un esquema dinámico (no todos los documentos almacenados tienen por qué tener los mismos campos). (MongoDB, s.f)

Como última aclaración, las tablas en Mongo pasan a llamarse “colecciones”, y cada fila de esas tablas, “documentos”.

PyMongo

PyMongo es una librería distribuida para Python que sirve como “wrapper” de MongoDB, es decir, gracias a ella se puede manejar la base de datos NoSQL directamente desde el código, sin tener que recurrir al cliente “mongo” de la terminal. (PyMongo documentatio, s.f)

Herramientas para el procesamiento del lenguaje natural

Una vez explicados los conceptos y las técnicas que se han usado a lo largo de este proyecto se pasarán a exponer las diferentes herramientas que ofrecen un procesamiento de lenguaje natural y que se valoraron para ser utilizadas.

Existe una gran variedad de herramientas que desempeñan esta función, aunque la mayoría suelen operar con el idioma inglés más que con el español, y además cada uno cuenta con sus propias características que lo diferencian bien del resto. Para elegir el que mejor se adecúa a las necesidades del proyecto, fue necesario analizar cada una de las alternativas para saber qué ofrece cada herramienta.

En general, las características que son generales a todas las herramientas de este tipo son:

- Análisis morfológico
- Etiquetado gramatical
- Segmentación de frases
- Tokenización y lematización
- Analizador sintáctico
- Analizador de dependencias
- Correferencias
- Grafo semántico

Además, a la hora de elegir la herramienta adecuada se ha tenido en cuenta los idiomas que cada una maneja, cosa importante ya que se pretenden utilizar textos escritos en español y no todas lo soportan.

NTLK

Natural Language Toolkit es una plataforma para el procesamiento del lenguaje natural para el lenguaje de programación Python. Proporciona interfaces fáciles de usar, con más de 50 corpus y recursos léxicos como WordNet, una de las mayores bases de datos léxicas para el inglés, así como un conjunto de bibliotecas de procesamiento de texto multilingüe. (Natural Language Processing with Python, s.f)

Es un módulo muy potente y completo. Cuenta con una serie de corpus y colecciones de texto de una gran variedad de idiomas, precargados e indexados, que facilitan el uso y agilizan en gran medida las pruebas. Así mismo, es una librería extensa, con múltiples métodos de cálculo de frecuencias y probabilidades, búsquedas de términos y de textos, analizadores de frases, extracción y segmentación de información, etc. En cuanto a las características de procesamiento, encontramos:

- Clasificación de textos
- Tokenización de texto en palabras
- Segmentación del texto en frases
- Análisis morfológico
- Etiquetado gramatical
- Lematización
- Análisis semántico

Tras haber realizado un conjunto de pruebas con textos en inglés, se observó que los resultados eran excelentes, pero no sucedió lo mismo con textos en español, por lo que se decidió descartarla.

Spaguetti Tagger

Esta librería está creada a partir de los recursos disponibles en la librería NTLK de Python para el tratamiento de textos en español. (Alvations, 28 septiembre 2016)

Sin embargo, el corpus que usa no permite identificar algunas de las palabras, por lo que se optó por descartar esta opción y seguir investigando, ya que no era muy viable.

TreeTagger Wrapper

Este módulo opera sobre veinte idiomas distintos, y además es adaptable a nuevos si se provee un lexicón y un corpus de texto adecuados. (TreeTagger Python Wrapper's Documentation, s.f)

Sin embargo, no posee tantas funcionalidades como la mayoría de este tipo de herramientas, pues solo ofrece lematización, etiquetado gramatical del texto, así como un stemming de palabras si se trabaja en inglés, alemán, francés o español.

Debido a las pocas funcionalidades que ofrece, se ha investigado esta herramienta únicamente con propósito académico.

Stanford CoreNLP

Desarrollada por la Universidad de Stanford. Está implementada en Java, aunque existen interfaces para otros lenguajes como Perl, Ruby o Python. (Using Stanford CoreNLP within other programming languages and packages, s.f)

Este módulo presenta varias características:

- Tokenización del texto en palabras
- Segmentación del texto en frases
- Etiquetado gramatical
- Lematización
- Reconocimiento de nombres de entidades (NER)
- Análisis semántico
- Grafo semántico

Sin embargo, se presenta otra vez un problema común en este tipo de herramientas y es el soporte de idiomas: esta herramienta está entrenada para textos en inglés, y lo que interesa en este proyecto es analizar textos en español.

Ixa Pipes

Se trata de un conjunto de módulos para el procesamiento del lenguaje natural creada por el grupo IXA NLP de la Universidad del País Vasco. Es bastante fácil de utilizar y soporta una variedad limitada de idiomas (IXA pipes: ready to use NLP tools, s.f). Sus características son:

- Tokenización del texto en palabras
- Segmentación del texto en frases
- Análisis morfológico
- Reconocimiento de nombres de entidades (NER)
- Chunker de palabras (solo vasco)
- Análisis sintáctico (solo español e inglés)

Posee varias funcionalidades a tener en cuenta, y lo resultados son buenos.

Freeling

Se trata de una librería implementada en C++ que ofrece varias herramientas para el procesamiento del lenguaje natural y de textos. Ha sido desarrollada por la Universitat Politècnica de Catalunya (UPC) bajo licencia de Software Libre, y permite trabajar además sobre una variedad amplia de idiomas, entre ellos el español, alemán, inglés, francés, croata, etc. (Freeling 4.1 User Manual, s.f)

Actualmente hay desarrollada una interfaz para Python, si bien su instalación es un poco tediosa. Las funcionalidades que posee son:

- Tokenización del texto en palabras
- Segmentación
- Análisis morfológico
- Etiquetado gramatical
- Lematización
- Detección de nombres de entidades (NER)
- Análisis sintáctico

- Analizador de dependencias y correferencias
- Grafo semántico

Como se puede observar, es una de las pocas herramientas (si no la única) que posee una amplia variedad de características, aunque hay algunas que no están disponibles para todos los idiomas. Sin embargo, otra de las ventajas que tiene es que el idioma central sobre el que está desarrollado, y para el que mejor preparado está, es el español con el cual todas las funcionalidades listadas se pueden usar sin problemas.

Observaciones finales

Después de haber estudiado cada una de las opciones y realizar ciertas pruebas para ver cuál se desenvuelve mejor en la tarea de este proyecto, se ha decidido que la herramienta a usar será Freeling.

La principal razón es que la mayoría de las herramientas están disponibles o bien solo para el idioma inglés o bien soportan de manera muy básica el español, por lo que los resultados no son tan buenos como se pueden esperar. Sin embargo, como ya se ha comentado, Freeling está programado en base al español, por lo que sus resultados son más que correctos para cumplir los objetivos.

Gensim

Se trata de una librería para Python que realiza modelado semántico no supervisado y es capaz de crear así modelos de lenguaje a partir de un entrenamiento previo usando un corpus extenso (en nuestro caso, los artículos de la versión de la Wikipedia en español en el año 2014 lematizados). Gensim implementa diferentes técnicas para obtener la modelización semántica de los textos tratados, por ejemplo: Análisis de Semántica Latente, Asignación de Dirichlet Latente, y también incluye modelos Word2Vec. (Gensim: Topic modelling for humans, s.f)

Sci-Kit Learn

Se trata de una librería de aprendizaje automático para Python que pone a disposición del usuario varios algoritmos de clasificación y regresión incluyendo support vector machines (SVMs) (Scikit-Learn: Machine Learning in Python, s.f). En este proyecto, se ha usado el algoritmo de clasificación Support Vector Machine (máquina de vectores de soporte).

Así mismo, la librería ofrece métodos para separar el conjunto original de datos en dos subconjuntos: uno de entrenamiento y otro de prueba, con la posibilidad de indicar la proporción de cada uno con respecto al original. También se dispone en esta librería de métodos para validación cruzada.

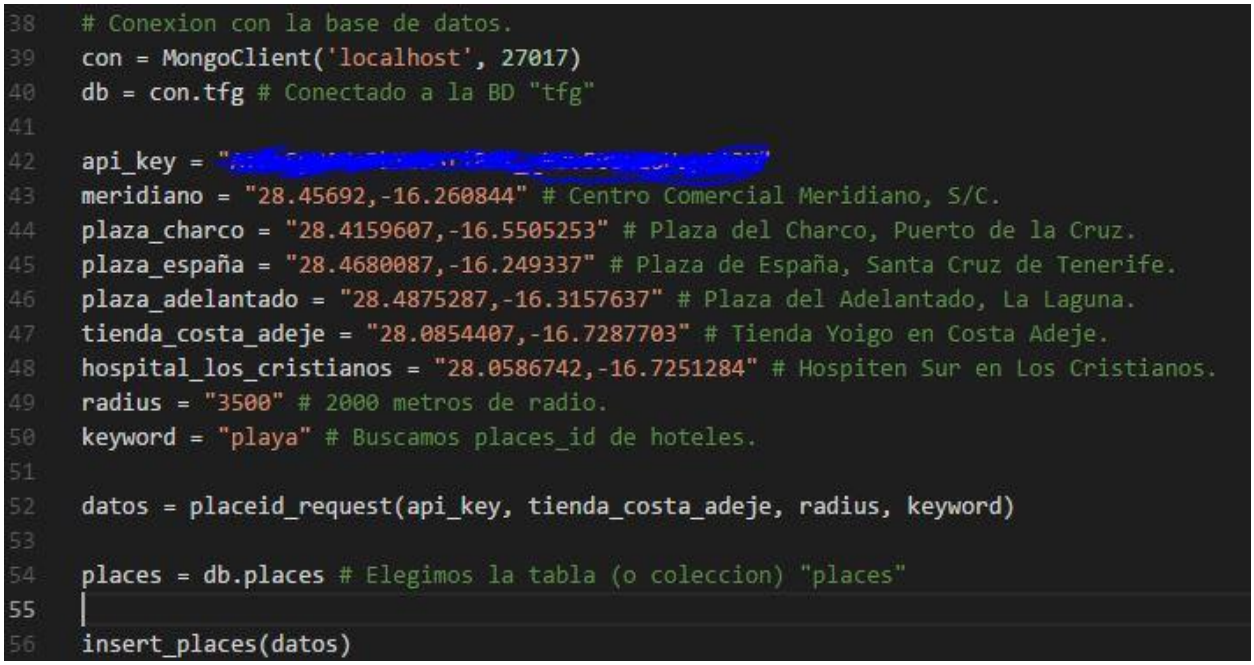
Capítulo 3 – Diseño e implementación

Tras realizar el estudio previo a las tecnologías que se van a usar, se pasa a la siguiente fase del proyecto: la implementación de la estructura de procesamiento de lenguaje natural y análisis de sentimientos. Esta fase ha ocupado la mayor parte del tiempo invertido en el proyecto, puesto que ha habido que experimentar sobre diversas técnicas que o no han funcionado del todo, o bien han tenido que ser mejoradas. A continuación, se detallarán todos los pasos llevados a cabo para construir la infraestructura de prueba y alcanzar los objetivos.

Obtención de los sitios

Para poder pasar a tratar la información, primero es necesario obtenerla. Dado que lo que interesa en este proyecto son opiniones escritas por los usuarios sobre distintos lugares con interés turístico, se ha decidido usar Google Maps como principal fuente de datos (aunque se podrían haber usado otras). La elección de esta fuente se debe a la facilidad que ofrece su API para recoger la información que nos interesa, es decir: opiniones, nombre del lugar, dirección… así como datos que puedan ser usados para posibles expansiones de la plataforma como geolocalización, horario de apertura, etc.

Como se ha comentado en el estudio previo, para el uso de la API de Google Maps es necesario solicitar una clave que proporciona la plataforma Google Cloud Console para lo que es necesario registrarse y crear un proyecto. Esta se usa en el script: **places.py,** que recoge los sitios de interés, aquellos de los que se van a recabar opiniones.



```
38  # Conexion con la base de datos.
39  con = MongoClient('localhost', 27017)
40  db = con.tfg # Conectado a la BD "tfg"
41
42  api_key = "[redacted]"
43  meridiano = "28.45692,-16.260844" # Centro Comercial Meridiano, S/C.
44  plaza_charco = "28.4159607,-16.5505253" # Plaza del Charco, Puerto de la Cruz.
45  plaza_españa = "28.4680087,-16.249337" # Plaza de España, Santa Cruz de Tenerife.
46  plaza_adelantado = "28.4875287,-16.3157637" # Plaza del Adelantado, La Laguna.
47  tienda_costa_adeje = "28.0854407,-16.7287703" # Tienda Yoigo en Costa Adeje.
48  hospital_los_cristianos = "28.0586742,-16.7251284" # Hospiten Sur en Los Cristianos.
49  radius = "3500" # 2000 metros de radio.
50  keyword = "playa" # Buscamos places_id de hoteles.
51
52  datos = placeid_request(api_key, tienda_costa_adeje, radius, keyword)
53
54  places = db.places # Elegimos la tabla (o coleccion) "places"
55
56  insert_places(datos)
```

Figura 3

La Figura 3 corresponde al “main” del script **places.py**. Como podemos observar, es necesario usar la “api_key” (se ha ocultado en la imagen porque debe ser única para cada usuario y por tanto confidencial). Las otras variables declaradas son las coordenadas de los lugares que se usarán como punto central. Tomando este punto, la API devolverá los lugares en un radio de “radius” metros que coincidan con la keyword “keyword”. Las keywords que se han utilizado son: restaurante, playa, parque y hotel.

En la línea 52 se llama a una función que hace uso del módulo “requests” (explicado en el capítulo 3) que hace la petición a la API de Google Maps pasándole los parámetros mencionados.

Por último, en la línea 56 se insertan los datos en la base de datos MongoDB. En esta función se comprueba que el sitio no esté insertado ya en la base de datos, y si es así, se construye un diccionario con los siguientes campos que será lo que se inserte en Mongo:

- _id: campo de identificador requerido por Mongo.
- place_id: el identificador único de cada sitio devuelto por la API.
- name: nombre del sitio.
- JSON: el JSON crudo devuelto por la API con todos sus campos, a partir del

cual se construye cada campo de la colección.

- type: tipo de sitio, indicado por la keyword (restaurante, hotel, playa, parque).

Obtención de las opiniones

Una vez almacenados los sitios y su información, se procede a obtener las opiniones escritas por los usuarios de cada uno de ellos. Por defecto, la API de Google Maps, en cada petición, solo devuelve las últimas cinco opiniones (aunque ampliable con un plan de pago).

Es necesario mencionar, que para obtener las opiniones de Google Maps, la API indica que se haga la petición a otro enlace distinto en el que solo se indica el "place_id" del sitio y la "api_key". Por tanto, sabiendo esto, se desarrolló el script **reviews.py**. (Figura 4)

```
query = places.find({}, {'type': 1, 'name':1, 'place_id': 1, '_id': 0}) # SELECT places_id FROM places


for i in query:
    row = {}
    res = requests.get('https://maps.googleapis.com/maps/api/place/details/json?placeid='+i['place_id']+'&key='+api_key+'&language=es')
    json_data = res.json()

    if 'reviews' in json_data['result'].keys(): # Si el sitio tiene reviews
        for j in range(len(json_data['result']['reviews'])):
            resultados = reviews.find({'review': json_data['result']['reviews'][j]['text'], 'name': json_data['result']['name']}) # Hacemos un find a ve
            if(resultados.count() == 0): # Si la review NO está en la BD, insertamos
                row['name'] = i['name']
                row['type'] = i['type']
                if 'rating' in json_data['result'].keys():
                    row['avg_rating'] = json_data['result']['rating'] # Valoración media del sitio
                row['location'] = {'lat': json_data['result']['geometry']['location']['lat'], 'lng': json_data['result']['geometry']['location']['lng']}
                if 'language' in json_data['result']['reviews'][j].keys():
                    row['language'] = json_data['result']['reviews'][j]['language'] # Idioma de la review
                row['rating'] = json_data['result']['reviews'][j]['rating'] # Valoración individual de la review
                row['review'] = json_data['result']['reviews'][j]['text'] # Review
                row['_id'] = ObjectId()
                reviews.insert(row) #Insertamos en la BD
                print("Insertado\n")
            else:
                print("La review:\n '"+ json_data['result']['reviews'][j]['text']+"'\n ya está insertada")
```

Figura 4

La primera línea realiza un query a la base de datos para obtener las "place_id" de cada sitio (recordemos que este dato es importante para obtener las opiniones).

Tras esto, iteramos sobre cada una de las "place_id" y realizamos la petición a la API, que nos devuelve un JSON con toda la información relativa a ese sitio: el nombre, dirección, horario, localización, opiniones, etc.

Es necesario comprobar que el sitio en cuestión posea opiniones, ya que si no el JSON devuelto no tendrá dicho campo y todo el script fallará. También se realiza la

comprobación de que la opinión no esté ya insertada en la base de datos, tras lo cual se procede a crear el diccionario con todos los campos que querremos almacenar en la colección “reviews” de la base de datos:

- name: nombre del sitio.
- type: tipo de sitio.
- avg_rating: valoración media del sitio (en base a todas las opiniones). Entre 1 y 5.
- location: coordenadas del lugar.
- language: idioma de la opinión.
- rating: valoración individual de la opinión. Entre 1 y 5.
- review: la opinión del sitio.

Después de esto, se almacena el diccionario como una fila (o documento) de la colección. Cabe mencionar que, por motivos de facilidad a la hora de recuperar cada opinión más adelante, se procedió al almacenamiento siguiendo el “esquema” (cada etiqueta entre llaves hace referencia a un campo del documento):

- Fila 1: {nombre 1}, {opinión 1}.
- Fila 2: {nombre 1}, {opinión 2}.
- Fila 3: {nombre 1}, {opinión 3}.
- …

Esto facilita la comprobación necesaria respecto a la existencia de una opinión determinada en la base de datos, frente al “esquema”:

- Fila 1: {nombre 1}, {opinión 1, opinión 2, opinión 3, opinión 4, opinión 5}
- Fila 2: {nombre 2}, {opinión 1, opinión 2, opinión 3, opinión 4, opinión 5}
- …

De esta segunda forma, se debería recorrer el array comprobando que la opinión que se va a insertar no esté ya introducida, y si no lo está, crear un array de copia, introducir la nueva opinión y actualizar el campo mediante la función “update()” de MongoDB. Sin embargo, en la primera versión solo hay que listar todas las opiniones

introducidas, comprobar que no esté la que se quiera insertar, y si es así crear un nuevo diccionario que será el documento a añadir en la colección.

Cuando el script finalice su ejecución, se tendrá una nueva colección en Mongo llamada “reviews” que tendrá almacenadas todas las opiniones de un sitio determinado.

Lematización de las opiniones

Es en este paso del desarrollo del proyecto donde se usa la herramienta de procesamiento de lenguaje natural Freeling. Para el propósito de nuestro proyecto, únicamente nos interesa la funcionalidad de la lematización de palabras.

Freeling requiere cargar sus módulos previamente. Es importante destacar que la herramienta se carga en función del idioma que se vaya a utilizar. Por tanto, en el caso de querer analizar opiniones de distintos idiomas, habría que cargar Freeling cada vez para poder cargarlo en el idioma que se requiera. En un principio se probó a analizar opiniones en distintos idiomas (ya que Freeling posee un detector de idioma para un texto determinado), pero se observó que el tiempo de carga de Freeling cada vez que se analizaban distintas opiniones, era demasiado alto. Dado que se salía del alcance del proyecto obtener una herramienta totalmente funcional, se optó por analizar opiniones únicamente en español, ya que el procedimiento será el mismo para los idiomas que se quieran agregar y que estén disponibles.

Una vez cargada la librería de Freeling, se realiza una consulta a la colección “reviews” para obtener todos los nombres y la información de sitios que tengan opiniones (recordemos que no todos las tenían). Una vez tengamos todos los datos, se almacenarán en un diccionario el nombre, la valoración, la opinión original (es decir, sin lematizar) y se pasará a analizar cada una de ellas.

```
def freeling_analysis(lin, p_a_f_e, p_a_s): #Metodo que recibe la review, y la analiza con freeling para obtener la FC y la etiqueta.

    while(lin): #Mientras no se acabe el texto introducido
        l = tk.tokenize(lin) # Tokenizamos
        ls = sp.split(sid, l, True) # Dividimos la cadena. "True" indica que se dividan cadenas incluso si no acaban en punto.
        ls = mf.analyze(ls) # Analizador morfológico
        ls = tg.analyze(ls) #Permite obtener entidades (lugares, nombres propios, etc)
        global parsed_string

        for s in ls:
            ws = s.get_words()
            for w in ws:

                parsed_string += w.get_lemma() + " "
                #print(p_s)
                #print("\n")
                p_a_s.append(w.get_lemma())
                #print(p_a_s)
                #print("\n")

                p_a_f_e.append({'FC': w.get_lemma(), 'Etiqueta': w.get_tag()})
                #print(p_a_f_e)
                #print("\n")


        break
```

Figura 5

El análisis con Freeling pasa por varias fases. Cada una de ellas se realiza dentro de un bucle while() que itera hasta que se acabe de analizar la cadena. Dentro del bucle se comienzan a ejecutar las distintas fases de la lematización.

Clase “word”

Es necesario comenzar describiendo esta clase, ya que es la clase básica de toda la herramienta y la que usa en casi todas sus funcionalidades, almacenando la forma de una palabra, e información relativa a ella como su codificación fonética.

Debido a la ambigüedad presente en algunas palabras (por ejemplo, “guapo” puede ser sustantivo o adjetivo), es necesario también guardar una lista de objetos “analysis”, que están formados por la forma lematizada de la palabra en sí y su etiqueta Part-of-Speech (PoS, o lo que es lo mismo, su categoría gramatical). Estos objetos también pueden contener otra información como probabilidad, listado de sentidos, etc.

Tokenizador

El primer módulo que se aplica en el pipeline de procesamiento que sigue la herramienta es el tokenizador. Su función es convertir texto plano en un vector de objetos “word”, de acuerdo con unas reglas de tokenización basadas en expresiones regulares.

Splitter

El siguiente proceso se encarga de obtener las frases en la cadena. Para ello se utiliza este módulo que recibe una lista de objetos "word", y los almacena en búffer hasta que un final de frase es detectado, retornando así una lista de objetos "sentence".

En la captura anterior se puede observar la llamada a la función "split" a través del objeto "sp". Es importante destacar aquí el parámetro booleano que se le pasa a dicha función, pues determinará cuándo se devolverá la lista de objetos "sentence". Esto se ve mejor con un ejemplo. Considérese el siguiente texto:

Buen sitio, el trato fue muy amable. El mejor hotel al que he ido

Como se puede ver, no acaba en punto final, por lo que si el booleano está a "False" el splitter quedará a la espera de lo que venga a continuación, y el objeto *sentence* solo contendrá "Buen sitio, el trato fue muy amable." Sin embargo, si el booleano está a "True", se forzará el vaciado del buffer aunque no se haya encontrado un punto y final y por tanto se devolverá el texto completo como un objeto *sentence*.

Analizador morfológico

Se trata de un meta-módulo que no realiza un procesamiento por sí mismo, sino que su declaración sirve para inicializar todos los submódulos que ofrece:

- Detección de puntuación.
- Detección de números.
- Módulo User-Map (asigna etiquetas PoS a palabras que casen con expresiones regulares).
- Detección de fechas.
- Módulo de búsqueda en diccionario (para encontrar formas lematizadas y etiquetas PoS).
- Módulo de reconocimiento de palabras múltiples ("a buenas horas", "a causa de", etc.)
- Reconocimiento de entidades (útil para nombres de lugares, personas, etc.)
- Reconocimiento de cantidades (como porcentajes, fracciones, etc.)
- Módulo de asignación de probabilidades y adivino de palabras desconocidas (asigna una probabilidad a priori, y si la palabra no ha sido analizada con éxito con ningún otro submódulo, se intenta etiquetar basándose en el

final).

Así, una vez declarado este meta-módulo, se puede dar paso a analizar la frase. Cada resultado se puede recuperar con funciones del tipo “get()”. Para el caso del proyecto, **get_lemma()** y **get_tag()** eran las pertinentes. Con estos resultados, se creó para cada opinión analizada:

- Una string (en el script **parsed_string**) que contiene la oración en forma lematizada.

  Bueno sitio, el trato ser muy amable. El mejor hotel a el que haber ir.

- Un array que contiene las palabras lematizadas.

  [“Bueno”, “sitio”, “,”, “el”, “trato”, “ser”, …]

- Un array de diccionarios con la palabra lematizada y su etiqueta.

  [{‘FC’: “Bueno”, ‘Etiqueta’: “A..”}, …]

La elección de guardar de estas tres formas la cadena analizada se debía a que no se sabía en un principio en qué formato resultaba mejor pasarle la cadena lematizada a la herramienta de aprendizaje automático. Además, también se pensó en la escabilidad de la plataforma, dado que almacenar los resultados de distintas formas puede resultar beneficioso y ahorrar tiempo en un futuro al usar otras herramientas.

Por último, el documento que contiene estos tres resultados más la opinión original, el nombre del sitio y la valoración, es insertado en la base de datos.

Word2Vec

Para esta fase del desarrollo del proyecto, se ha utilizado la librería que implementa los algoritmos de Word2Vec para Python, Gensim. Se han llevado a cabo dos etapas: entrenamiento de un modelo y obtención del vector Word2Vec medio.

Entrenamiento del modelo

Es necesaria una etapa de aprendizaje del algoritmo. Para ello, se usará un corpus formado por los artículos de la Wikipedia en español, pero en forma lematizada (esto es importante dado que las opiniones que se analicen estarán en forma lematizada como se vio en el apartado anterior). Se considera la Wikipedia como un corpus lo suficientemente amplio como para tener una muestra del uso del lenguaje a partir del que inferir la representación numérica de las palabras. Esto creará un

vocabulario de palabras con el cual el algoritmo tendrá una referencia de qué palabras puede convertir a vector numérico y cuáles no (gracias al entrenamiento previo).

```
from gensim.models.word2vec import LineSentence
from gensim.models import Word2Vec

# Creamos array con cada linea del corpus
sentences = LineSentence('wikipedia_lematizada.txt')

min_count = 4 #Numero minimo de veces que tiene que aparecer una palabra para no ser descartada
size = 300 # Dimension del vector resultante
window = 5 # Se usaran "window" palabras para denotar el contexto
workers = 4 # Numero de hilos.

print("Creando vocabulario...")
model = Word2Vec(sentences, min_count=min_count, size=size, window=window, workers=workers)
print("Entrenando modelo...")
model.train(sentences, total_examples=model.corpus_count, epochs=10)
model.save('modelo_entrenado2.txt')
print("Modelo entrenado y salvado.")
```

Figura 6

La primera línea de la Figura 6 crea un array con cada línea del corpus. Esto es necesario debido a que la función que se encarga de entrenar el modelo necesita recibir el corpus en ese formato. Los parámetros que se usarán en la función que crea el vocabulario son los siguientes:

- min_count: es el número mínimo de veces que tiene que aparecer una palabra en el corpus para no ser descartada y añadirla al vocabulario. El tiempo de entrenamiento se incrementa exponencialmente cuanto más pequeño es este número, ya que se considerarán más casos.
- size: dimensión del vector numérico resultante. Cuanto más grande, más exactitud, pero también más tiempo de entrenamiento. Generalmente se utiliza un valor superior a 100, en nuestro caso se ha utilizado el valor 200.
- window: este valor denota el contexto que se usará, es decir, el número de palabras circundantes. Por ejemplo, con contexto 5 las palabras circundantes a “muy” serían las siguientes:

  Buen sitio, el trato fue muy amable. El mejor hotel al que he ido

- workers: el número de hilos que se crearán paralelamente para entrenar el modelo.

Definidos estos valores, se procede a la creación del vocabulario con la función “Word2Vec” a la que se le pasan los parámetros definidos anteriormente. Tanto esta función como la de “train” requieren un tiempo extenso de ejecución, aunque

depende de los parámetros usados. En este proyecto, aunque en la captura se refleje lo contrario, se ha usado:

- min_count = 5
- size = 200
- workers = 4
- window = 4

Se han probado otras combinaciones de parámetros, pero el tiempo de entrenamiento asciende casi exponencialmente, llegando a tardar más de un día en una máquina de cuatro núcleos y cuatro gigas de RAM. Por otra parte, cuando el entrenamiento finalizó, el modelo entrenado resultó ocupar demasiado y a la hora de cargarlo en RAM fue imposible ejecutar el script, por lo que los parámetros descritos han resultado los óptimos para este proyecto.

El modelo entrenado se almacena en un fichero para poder cargarlo más adelante. La librería Gensim ofrece la posibilidad de cargar modelos ya entrenados, así que se usa para ello dicha función.

Obtención del vector numérico medio

Esta etapa se realiza en otro script, **word2vec_mean.py** en el que cargaremos el modelo entrenado anteriormente para convertir cada opinión a un vector numérico.

```
18  print("Cargando modelo...")
19  model = gensim.models.Word2Vec.load("modelo_entrenado.txt")
20  print("Modelo cargado.")
21  word_vectors = model.wv
22
```

Figura 7

En la Figura 7 se aprecia el código que carga el modelo entrenado previamente y luego almacena en “word_vectors” todas las palabras en formato Word2Vec que posee el modelo en su vocabulario.

```
nombres = query_names() #Obtenemos los nombres de todos los sitios
rejected_words = []

for i in nombres:
    row = {}
    row['name'] = i #Nombre del sitio

    data = get_data(i) #Obtenemos todos los datos asociados a un sitio (reviews, tipo, etc.)
    for j in data:
        resultados = mean_word2vec.find({'original_review': j['original_review'], 'name': i})
        if(resultados.count() == 0): #Si la review no esta ya convertida a Word2Vec
            array_word2vec = []
            res = []
            media = []

            row['type'] = j['type']
            row['_id'] = ObjectId()
            row['original_review'] = j['original_review']
            row['rating'] = j['rating']

            s = j['parsed_string'].translate(str.maketrans('','',string.punctuation)) #Quitamos l
            ar = s.split() # Creamos una lista de strings separando cada palabra de la review
            for i in range(len(ar)):
                if ar[i] in word_vectors.vocab:
                    palabra = list(model.wv[ar[i]])
                    array_word2vec.append(palabra) #Tenemos matriz con cada palabra convertida a
                else:
                    rejected_words.append(ar[i]) #Recolectamos las palabras que no se encuentren

            '''
            print("Word2Vec:\n")
            print(array_word2vec)
            print("\n")
            '''
            if not array_word2vec:
                print("No se ha obtenido media de esta review. Quizá ninguna palabra estuviese en
            else:
                media = numpy.mean(array_word2vec, axis=0)
                binary_media = Binary(pickle.dumps(media, protocol=2), subtype=128) #Convertimos
                row['mean_vector'] = binary_media
                #print(row)
                mean_word2vec.insert(row)
                print("Inserted!\n")
        else:
            print("Already inserted!")

print("Palabras no aceptadas:")
print(rejected_words)
```

Figura 8

En la Figura 8 recuperan todas las opiniones lematizadas de cada sitio, se comprueba que la review no se ha convertido ya a vector numérico y que no se ha insertado. Si es así, se procede a:

- Añadir los campos “type” (tipo de sitio), “_id”, “original_review” y “rating” al documento que posteriormente será insertado en Mongo.
- Para cada review lematizada, se le quitan los signos de puntuación, paréntesis, llaves, corchetes y símbolos de divisa (€, $) que tenga.
- Se crea una lista de strings separando cada palabra de la opinión.
- Para cada palabra de esa lista:

    - Comprobamos que la palabra en cuestión esté en el vocabulario del modelo, y si no, la añadimos a una lista de palabras no encontradas (fenómeno que ha ocurrido 698 veces en palabras mal escritas o irreconocibles para Freeling como “apesardetodo”, “muuuy”, etc.).
    - Si la palabra está en el vocabulario, añadimos el vector numérico a una lista, a fin de que al final, cuando estén todas las palabras de la opinión añadidas, tengamos una matriz de tamaño n_palabras × 200 (dado que ésta era la dimensión del vector numérico).
- Si resulta que ninguna de las palabras de la opinión está en el vocabulario la matriz estará vacía y por tanto no nos interesa seguir. Es en este punto donde la ortografía, sintaxis e idioma de la opinión puede afectar drásticamente a los resultados.
- Si por el contrario, la matriz tiene elementos, se calcula la media de cada elemento por columnas de la siguiente forma:

Matriz: [[1,2,3], [2,1,3], [3,2,1]]

Media de la matriz: [[2, 1.6, 2.3]]

- Por último, la matriz resultante es de tipo array de la librería NumPy. Mongo no reconoce el tipo de dato, por lo tanto no permite insertarlo y es necesario convertirla a binario (línea 76).
- Repetir el proceso para todas las opiniones.

Clasificación de las opiniones

Una vez ejecutado el script anterior, lo que se obtendrá será un vector numérico medio que representa a cada opinión, del estilo:

[0.1321, 1.2344, -1.2323, 4.2342, …]

Este array corresponde al vector de características con el que será tratada por los algoritmos de aprendizaje automático. En este trabajo se ha elegido utilizar el algoritmo de aprendizaje SVM. El conjunto de entrenamiento está formado por vectores Word2Vec y la clasificación que se quiere aprender la obtenemos de las valoraciones numéricas de esa opinión que el usuario aportó, siendo los valores posibles: 1, 2 3, 4 y 5

El modelo va fijándose en cada uno de los arrays y las valoraciones que les corresponde: si al vector anterior que se puso de ejemplo se le asocia una meta (o

valoración) de 4, el algoritmo acabará aprendiendo que los vectores que tengan características parecidas son más propensos a tener esa puntuación.

Analizando el script classification.py

```
17  vector_rating = [] #Vector de diccionarios con la review y su rating
18
19  resultados = mean_word2vec.find({}, {'_id': 0, 'mean_vector': 1, 'rating': 1}) #Obtenemos los vectores y sus valoraciones.
20
21  for i in resultados:
22      vector_rating.append({'vector': list(pickle.loads(i['mean_vector'])), 'rating': i['rating']})
23
24  #Creamos dos listas: una lista de entrenamiento con el 80% de las reviews, y otra de testeo con el 20% restante.
25  samples = train_test_split(vector_rating, test_size=0.2, train_size=0.8, shuffle=False)
26
27  train_sample = samples[0]
28  test_sample = samples[1]
29
30  train_sample_vector = []
31  train_sample_rating = []
32
33  test_sample_vector = []
34  test_sample_expected_ratings = []
35
36
37  for i in range(len(train_sample)):
38      train_sample_vector.append(train_sample[i]['vector']) #Meter en una matriz todos los vectores
39
40  for i in range(len(train_sample)):
41      train_sample_rating.append(train_sample[i]['rating']) #Meter en una lista todas las ratings
42
43  for i in range(len(test_sample)):
44      test_sample_vector.append(test_sample[i]['vector']) #Meter en una matriz todos los vectores
45
46  for i in range(len(test_sample)):
47      test_sample_expected_ratings.append(test_sample[i]['rating']) #Meter en una lista todas las ratings esperadas.
48
49  #Así, ahora tenemos una lista de listas de tamaño n_muestras x n_características, que nos sirve como entrenamiento.
50
51  clf = svm.SVC(gamma=0.001, C=100.)
52  clf.fit(train_sample_vector, train_sample_rating) #Entrenamos con las reviews y sus valoraciones
53
54  predicted_ratings = clf.predict(test_sample_vector)
55
56  print("Valoraciones esperadas:")
57  print(test_sample_expected_ratings)
58  print("Valoraciones obtenidas con Scikit:")
59  print(predicted_ratings)
```

Figura 9

Las primeras líneas de la Figura 9 se encargan de recoger de Mongo todas las opiniones en formato Word2Vec medio (y convertirlas otra vez a tipo array en la línea 22) junto a sus valoraciones individuales usando los diccionarios de Python.

En la línea 25 se usa una función propia de la librería Sci-Kit para dividir una lista en dos subconjuntos: uno de prueba y otro de entrenamiento, pudiendo además indicarle la proporción con respecto a la lista original. Así, para este caso se le ha indicado que el subconjunto de entrenamiento sea el 80% de la lista de opiniones original, y el 20% sea el subconjunto de pruebas.

Tras hacer esta división, dividimos los datos en dos listas, una para los arrays

Word2Vec y otra para las valoraciones individuales (esta última lista solo se hace para el conjunto de entrenamiento). Al final, en la lista de los arrays, volvemos a tener una matriz de n_muestras (opiniones) * n_caracteristicas (en este caso 200, debido al parámetro “size” de cuando se entrenó el modelo del lenguaje).

La línea 51 corresponde a la declaración del objeto svm (support vector machines o máquinas de vectores de soporte). Mediante el entrenamiento (el cual se realiza en la línea 52) se puede construir un modelo que prediga la clase (en nuestro caso la valoración numérica) de una nueva muestra.

Tras el entrenamiento, se almacena en una lista vacía los resultados de las predicciones del algoritmo para luego mostrarlos.

Al ejecutar el script, se obtienen los resultados en la Figura 10:

```
Valoraciones esperadas:
[4, 4, 1, 3, 5, 3, 4, 3, 5, 3, 2, 4, 1, 5, 5, 5, 5, 5, 5, 5, 4, 4, 5, 5, 5, 5, 5, 4, 5, 4, 5, 5, 5, 5, 4, 4, 5, 1, 4, 3, 5, 3
, 5, 4, 4, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 3, 5, 5, 5, 4, 5, 5, 5, 5, 4, 2, 5, 5, 3, 3, 4, 3, 5, 5, 3, 5, 2, 5, 4, 1, 4, 4, 3,
1, 1, 5, 5, 4, 5, 5, 5, 5, 5, 3, 5, 3, 4, 5, 4, 5, 4, 5, 5, 5, 5, 3, 3, 5, 5, 4, 1, 4, 4, 4, 1, 5, 4, 4, 4, 5, 1, 2, 1, 4, 5,
 2, 1, 5, 1, 5, 4, 5, 4, 5, 5, 5, 4, 5, 4, 5, 5, 5, 5, 5, 5, 1, 5, 3, 5, 5, 5, 2, 4, 5, 4, 4, 5, 5, 5, 5, 5, 3, 5, 4, 1, 3, 4
, 4, 5, 4, 5, 5, 5, 5, 4, 5, 1, 5, 5, 5, 4, 5, 4, 5, 5]
Valoraciones obtenidas con Scikit:
[5 3 1 4 5 5 3 4 5 3 4 4 1 5 5 5 5 5 5 5 4 5 5 5 5 5 5 4 5 4 5 4 5 5 5 4 5
 1 5 5 5 4 5 5 4 5 5 5 5 4 1 5 5 5 5 5 1 5 5 5 4 5 5 5 5 5 4 5 5 5 4 4 2 5 5
 1 5 1 5 4 2 5 5 5 1 4 5 5 3 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 4 4 5 4 4 5 5 4 1
 4 5 4 1 5 4 5 4 5 1 1 5 4 5 4 4 5 1 5 4 3 4 5 5 5 4 5 5 5 5 5 5 5 5 1 4 5
 4 5 4 5 4 5 5 5 5 5 4 5 4 3 1 5 1 4 5 4 5 1 5 5 5 4 5 5 1 5 4 5 5 5 5 5 5]
Equal ratings:  120  out of 185.
```

Figura 10

Las valoraciones esperadas son las que los usuarios pusieron a la hora de escribir las opiniones que conforman el conjunto de pruebas. Las obtenidas con Scikit son las que la herramienta predijo.

Análisis de resultados

Tenemos que 120 de las 185 valoraciones de prueba coinciden. Sin embargo, el resultado no es tan exacto debido a que no se disponía de una cantidad suficiente de opiniones (a pesar de haber usado 924 opiniones en total). Para ir mejorando este resultado, se seguirá ejecutando todo el pipeline de scripts para recoger más opiniones. Es necesario tener en cuenta que algunas han perdido información debido a su descarte, bien porque estaban mal escritas o bien porque el Freeling es incapaz de analizarlas, y por tanto fueron descartadas; no es lo mismo una oración con diez palabras que con 5, algo de información se pierde y por tanto el entrenamiento pierde un poco de eficiencia.

También se hizo otro tipo de análisis mediante un gráfico. Se tomaron en cuenta

las diferencias entre las valoraciones esperadas y las obtenidas (sabiendo que se encuentran en el rango entre 1 y 5). Así, los posibles valores son 0 (si ambas son iguales), 1, 2, 3 y 4. Se consideran resultados buenos y aceptables cuando la diferencia está entre 0 y 2, mientras que los que estén entre 3 y 4 indican que los resultados no son precisos; no se considera el mismo tipo de fallo cuando se espera un 5 y se asigna un 4, que cuando se espera un 5 y se asigna un 0. Teniendo estos datos, se ha creado un histograma con la frecuencia de ocurrencia de dichas diferencias:

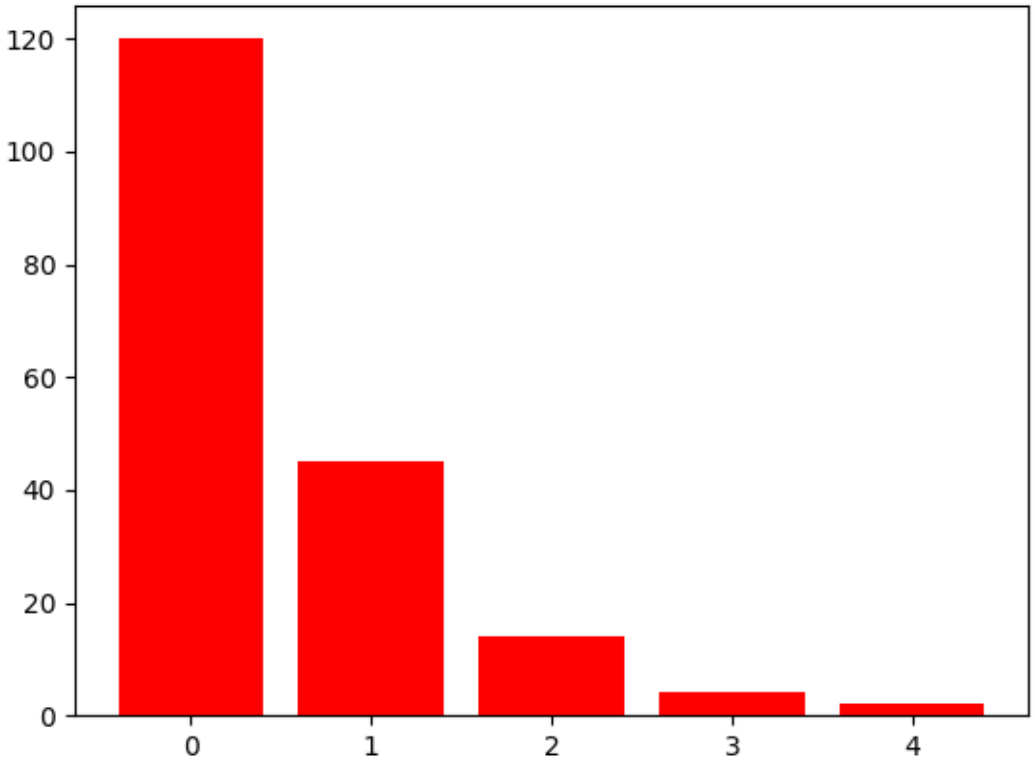


Figura 11

A partir de la Figura 11, se calcularán los porcentajes de frecuencia de cada valor:

Valores	Porcentaje de frecuencia
0	64,86%
1	24,32%
2	7,56%
3	2,16%
4	1,08%

Tabla 1: Valores frente a Porcentajes de frecuencia

Esto indica que en general el análisis ha sido muy bueno, aunque mejorable si no se hubiesen omitido algunas palabras como se comentó anteriormente.

Sin embargo, dada la debilidad que posee la metodología de la división de conjuntos y la diferencia entre resultados dependiendo de la separación entre datos de prueba y de entrenamiento, se decidió utilizar también la metodología de validación cruzada de K iteraciones.

A la hora de obtener resultados utilizando la validación cruzada, se realizaron varias pruebas modificando el número de iteraciones. Mientras que el estándar de esta cantidad suele ser 10, no solo se ha probado con dicho número, sino que también se ha probado con 5, 20 y 30 iteraciones.

Nº iteraciones	Media de los resultados
5	0.57 ± 0.08 (57%)
10	0.57 ± 0.10 (57%)
20	0.59 ± 0.19 (59%)
30	0.58 ± 0.16 (58%)

Tabla 2: Nº iteraciones frente a resultados

En general, el porcentaje de acierto es bajo, debido sobre todo a la cantidad de palabras eliminadas a la hora de la conversión a formato Word2Vec. Sin embargo, es un porcentaje que está por encima de la mitad, por lo que aunque sea mejorable, significa que para más de la mitad de las pruebas que se realizaron, se obtuvieron los resultados correctos.

El número de iteraciones no ha podido sobrepasar las treinta debido a que el propio Scikit-Learn avisa mediante un “warning” de que la clase menos poblada sólo tiene treinta y cinco miembros, y que este número no debe ser menor que la “K” que el usuario especifique. Aun así, a partir de las 30 iteraciones se comienza a observar un descenso en la precisión.

Conclusiones y líneas futuras

El procesamiento de lenguaje natural constituye un conjunto de técnicas que poco a poco van progresando y aumentando en número con el paso del tiempo. Su uso se extiende desde investigaciones hasta redes sociales, y todo ello ayuda a que cada vez se vaya dando soporte a más y más idiomas, o incluso mejorar el rendimiento con los idiomas ya soportados.

Es un campo que actualmente está en constante progreso y al cual le queda aún mucho por recorrer, pues el lenguaje se puede encontrar de diversas maneras y la más difícil es su forma coloquial o mal expresada. Sin embargo, no puede pasar desapercibido el esfuerzo que la comunidad de programadores y matemáticos está haciendo para crear herramientas como las que se han usado en este proyecto y que reflejan los grandes avances que se han hecho en módulos como el analizador sintáctico y morfológico.

Otro campo que tampoco se puede dejar sin mencionar es el de machine learning en su uso en tecnologías Big Data. Actualmente se está viendo un crecimiento cada vez más grande de las tecnologías de este tipo, y se pueden encontrar en cualquier lugar (sin ir más lejos, los recomendadores de productos que aparecen en páginas web como Amazon). Todo esto se logra mediante un aprendizaje de patrones que realiza el usuario: qué productos o páginas visita o le interesan y cuáles no, cuándo, horarios, etc. Y todo es procesado con algoritmos de aprendizaje automático capaces de aprender estos patrones para luego poder predecir.

No es muy descabellado pensar que en poco tiempo existirán más herramientas enfocadas a este ámbito de la inteligencia artificial (y más con el regreso de las redes neuronales) y que muchos más lenguajes se verán con soporte en ellas, ya que por mucho que el inglés esté globalizado, siempre existirán casos en los que este idioma no lo resuelva todo.

Tras el desarrollo del trabajo se han conseguido diversos logros: se ha logrado manejar con cierta soltura herramientas dedicadas a este campo como son las herramientas de aprendizaje automático de las cuales al principio no se poseía ningún conocimiento, se ha conseguido implementar un flujo de trabajo a través de los diversos scripts que se han ido explicando a lo largo de este documento, y que aun teniendo una pequeña cantidad de datos, se ha conseguido el objetivo principal

de este trabajo, si bien es mejorable recopilando una mayor cantidad de datos. Esto último es solucionable utilizando más almacenamiento o pagando una cuota mensual a Google para que la API no devuelva únicamente las últimas cinco opiniones.

A la hora de realizar trabajos futuros se tratarán temas que no se han abordado en este proyecto debido a que pertenecen a un proyecto completo. El tratamiento de la jerga específica en redes sociales (correcciones de palabras, sintaxis, la adecuación de la plataforma para que esta sea capaz de identificar cualquier término mal escrito o mal empleado en el lenguaje) se presenta como uno de los principales aspectos que se deben abordar en un proyecto futuro de estas características pues es de los textos escritos por usuarios de donde se saca la información necesaria. En este proyecto no se ha abordado la solución a este problema principalmente debido a una cuestión de tiempo, pues desarrollar dicha solución da para realizar otro proyecto de la misma magnitud que éste.

A nivel personal, me ha gustado el tema de este proyecto debido a que aspectos como la metodología de trabajo o las herramientas utilizadas era algo desconocido para mí, y gracias a este trabajo he podido iniciarme en un ámbito de la informática, y en concreto de la inteligencia artificial, que no conocía, pero del que seguro que no dejaré de aprender.

Summary and Conclusions

Natural language processing is formed by a set of techniques that keep progressing and growing in number as the time goes by. It is used not only in investigations but also in social networks, and this is very helpful as more languages are going to be supported, or performance with the current languages supported is going to be improved.

It is an area that nowadays is in constant improvement but is not finished at all, because it is possible to find language in many written ways and the harder one is that which is misspelled. However, it cannot be unmentioned the amount of effort the community of programmers and mathematicians is making to release tools and modules like the ones used in this project, and which show the many improvements that have been made in modules like the morphologic or parsing.

Another area that cannot be left unmentioned is machine learning and its use in Big Data technologies. These can now be found almost everywhere on the net (for example, product adviser which are used on webpages like Amazon). All of this is achieved through the machine learning of user patterns: which products or webpages does the user visit, which not, when, etc. Everything is processed with machine learning algorithms able to learn those patterns and then predict.

It is not strange to think about the release in a matter of years of new tools focused on this area of artificial intelligence (even more with the comeback of artificial neural networks) and many more languages which will be supported despite English being the most globalized one, because there will be sometimes when this language will not solve the problem.

After finishing the development of the project, many achievements have been earned: use with certain ease many tools dedicated to this area such as machine learning ones, of which I did not know anything about them at the very beginning. Another one is that a working pipeline have been implemented using the scripts which have been described throughout this document. The main objective of this work has also been achieved, though having a little quantity of data, which can be improved collecting a bigger amount of data. The solution for this is either use more storage capacity or pay Google’s fee to get more than the last five reviews.

When doing future projects, certain issues which have not been addressed in this project because they belong to a complete project will be addressed. Word mistakes and how to correct them, understanding the syntax of every review a user writes in social networks, identifying misspelled words or misused… These aspects are among the ones that must be correctly implemented in future projects, as user reviews are the main source of data for this kind of matter. A solution for this issue has not been addressed in this project mainly due to the time it takes to be developed, as implementing this solution is like carrying out another project like this one.

About myself, I really enjoyed doing this project because I did not know about some aspects such as the working methodology or the tools used, and thanks to this project I could start from scratch in this artificial intelligence area which I did not know but I am sure I will not stop learning about it.

Capítulo 4 – Presupuesto

Tiempos de ejecución del proyecto

Tipos	Descripción
Estudio previo	42 horas
Tiempo de desarrollo	50 horas

Tabla 3: Resumen de tiempos

Presupuesto

Características	Coste aproximado
Hora de trabajo	5,00€ (aproximación en base a proyectos similares de estas características) (José Gregorio Mesa Reyes, 15 febrero 2017)
PC (máquina virtual 4 núcleos, 8 GB de RAM)	200€ (en base al coste medio de los componentes)
PC (IaaS máquina virtual 4 núcleos, 4 GB de RAM)	100€ (en base al coste medio de los componentes).
Licencias	0€ (todas las herramientas usadas se distribuyen bajo Software Libre).

Tabla 4: Resumen de presupuestos

Repositorio de código

Se ha habilitado un repositorio GitHub al que se ha subido el código: https://github.com/nantesfer/TFG

Referencias

- Álvaro Gutiérrez. (22 julio, 2015). El 75% de los viajeros planifican de forma online sus vacaciones. Ecommerce-news. Recuperado de https://ecommerce-news.es/el-75-de-los-viajeros-planifican-de-forma-online-sus-vacaciones-29339
- Big Data. Retos y oportunidades para el tursimo. (Sin fecha). Thinktur. Recuperado de http://www.thinktur.org/media/Big-Data.-Retos-y-oportunidades-para-el-turismo.pdf
- José Gregorio Mesa Reyes. (15 febrero, 2017). Análisis de hostelería. GitHub. Recuperado de https://github.com/jgmesareyes/HostelryAnalysis
- Word embedding. (Sin fecha). En Wikipedia. Recuperado el 3 de agosto de 2018 de https://en.wikipedia.org/wiki/Word_embedding
- Word2Vec. (Sin fecha). En Wikipedia. Recuperado el 3 de agosto de 2018 de https://en.wikipedia.org/wiki/Word2vec
- Análisis de sentimiento, ¿qué es, cómo funciona y para qué sirve?. (19 julio, 2017). Intelligent. Recuperado de https://www.itelligent.es/es/analisis-de-sentimiento/
- Máquinas de vector de soporte. (Sin fecha). En Wikipedia. Recuperado el 3 de agosto de 2018 de https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte
- Python. (Sin fecha). En Wikipedia. Recuperado el 3 de agosto de 2018 de https://es.wikipedia.org/wiki/Python
- Pip. (Sin fecha). En Wikipedia. Recuperado el 3 de agosto de 2018 de https://es.wikipedia.org/wiki/Pip_(administrador_de_paquetes)
- Get API Key. (Sin fecha). Google Developers. Recuperado de https://developers.google.com/places/web-service/get-api-key?hl=es
- Place details. (Sin fecha). Google Developers. Recuperado de https://developers.google.com/places/web-service/details?hl=es
- Requests: HTTP for humans. (Sin fecha). Python-Requests. Recuperado de

http://docs.python-requests.org/en/master/

- MongoDB. (Sin fecha). En Wikipedia. Recuperado el 3 de agosto de 2018 de https://es.wikipedia.org/wiki/MongoDB
- PyMongo documentation. (Sin fecha). MongoDB API. Recuperado de https://api.mongodb.com/python/current/
- Natural Language Processing with Python. (Sin fecha). NTLK. Recuperado de http://www.nltk.org/book/
- Alvations. (28 septiembre 2016). Spaghetti-Tagger. GitHub. Recuperado de https://github.com/alvations/spaghetti-tagger
- TreeTagger Python Wrapper's Documentation. (Sin fecha). TreeTagger Wrapper. Recuperado de http://treetaggerwrapper.readthedocs.io/en/latest/
- Using Stanford CoreNLP within other programming languages and packages. (Sin fecha). StanfordNLP. Recuperado de https://stanfordnlp.github.io/CoreNLP/other-languages.html
- IXA pipes: ready to use NLP tools. (Sin fecha). IXA2 Edu. Recuperado de http://ixa2.si.ehu.es/ixa-pipes/
- Freeling 4.1 User Manual. (Sin fecha). TALP-UPC. Recuperado de https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/
- Gensim: Topic modelling for humans. (Sin fecha). Radim Rehurek. Recuperado de https://radimrehurek.com/gensim/
- Scikit-Learn: Machine Learning in Python. (Sin fecha). Scikit-Learn. Recuperado de http://scikit-learn.org/stable/
- Mikolov, T. S. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems.
- Yoshua Bengio, R. D. (2003). En *A Neural probabilistic language model. Journal of Machine Learning Research. V 3* (págs. 1137-1155).
- Giatsoglou, M. V. (2017). *Sentiment analysis leveraging emotions and word embeddings. Expert Systems with Applications.*
- Cross-validation: evaluating estimator performance. (Sin fecha). Scikit-Learn. Recuperado de http://scikit-learn.org/stable/modules/cross_validation.html