Proyecto Fin de Grado
Ingeniería de Telecomunicación

Análisis de opinión mediante servicios de procesamiento de lenguaje natural con Amazon Comprehend de AWS

Autor: Antonio Manuel Romera Rodriguez
Tutor: Antonio Jesús Sierra Collado
Cotutor: Álvaro Martín Rodríguez

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Grado
Ingeniería de Telecomunicación

Análisis de opinión mediante servicios de procesamiento de lenguaje natural con Amazon Comprehend de AWS

Autor:
Antonio Manuel Romera Rodriguez

Tutor:
Antonio Jesús Sierra Collado

Cotutor:
Álvaro Martín Rodríguez

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Grado: Análisis de opinión mediante servicios de procesamiento de lenguaje natural con Amazon Comprehend de AWS

Autor: Antonio Manuel Romera Rodriguez

Tutor: Antonio Jesús Sierra Collado
Cotutor: Álvaro Martín Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quiero agradecer a todos aquellos que han hecho posible la realización de este Trabajo Fin de Grado. A mi familia y mi pareja, que siempre me han apoyado y animado durante todo mi recorrido académico. A mis tutores, por todos sus consejos y ayuda durante la elaboración de este trabajo. Por último, a mis amigos, compañeros de trabajo, de biblioteca y de ocio con los que he tenido la suerte de compartir muchos buenos momentos.

Resumen

Hoy día nos encontramos en la era de la información, los datos se han convertido en un activo muy valioso. Es por esto por lo que cada vez son más las empresas que quieren sacar provecho a este activo. Dando lugar a grandes inversiones en el campo del *Machine Learning* tanto privadas como públicas.

Debido al creciente interés en el campo y las grandes inversiones de los últimos años han surgido productos enfocados al *Machine Learning*. Estos ponen al alcance de empresas y particulares no especializados en esta tecnología herramientas que les permiten explotar los datos de los que disponen.

Este trabajo se centra en estudiar las principales opciones que existen en la actualidad relacionadas con el *Machine Learning*. Nos centramos en aquellas que nos permiten analizar textos, concretamente, las de procesado del lenguaje natural (del inglés, NLP). Durante el estudio de mercado se analizan las opciones que nos ofrecen las tres principales empresas del sector de la computación en la nube: Amazon Web Services, Microsoft Azure y Google Cloud. Estudiamos las opciones que nos ofrecen las tres empresas mencionadas, eligiendo la que mejor cubre nuestras necesidades al menor precio.

Una vez realizado el estudio del mercado, hemos creado una aplicación para poner en práctica lo que nos ofrece estos productos y poder comparar las opciones de Amazon Web Services y la de Microsoft Azure.

La aplicación consiste en analizar los comentarios de distintos restaurantes, clasificándolos en función de dos parámetros: el idioma en el que están escrito y el sentimiento que transmiten. La fuente de los datos es Google Maps. Para realizar la prueba, hemos dividido el escenario en cuatro fases.

- En la fase inicial procedemos a recolectar la información. Para ello empleamos la API de Google Maps, concretamente *Place Details*, empleando los números de teléfono de los locales para localizarlos dentro de Maps.
- Durante la segunda fase analizamos los datos que obtenemos de la primera. Para esta fase empleamos los servicios de Amazon Web Services, concretamente Comprehend y los de Microsoft Azure, concretamente Text Analytics. En esta fase llevamos a cabo dos clasificaciones. La primera en función al idioma empleado en el comentario y la segunda en función a los sentimientos que transmiten dichos comentarios. Lo hacemos primero con la opción de AWS y, luego, con la de Azure.
- En la tercera fase procesamos los datos para crear estadística, para ello usamos los datos que obtenemos de la fase anterior. Contamos los comentarios que quedan en cada clasificación y creamos gráficas que emplearemos en la siguiente fase. El objetivo de esta fase es preparar los datos para poder visualizarlos de forma ágil.
- En la cuarta y última fase creamos un documento HTML en el que, a partir de una plantilla, rellenamos con los datos y las gráficas obtenidas en los apartados anteriores.

De esta forma tras terminar todas las fases, los datos han seguido el siguiente proceso:

Recopilación, análisis, tratamiento y visualización.

Tras finalizar las fases comentadas comparamos los resultados que hemos obtenido con ambos servicios.

Como conclusión podemos decir que ambos servicios tienen un comportamiento similar, si bien AWS parece "comprender" mejor el contexto de la oración que Azure, acercándose más a lo que podría hacer un ser humano. Esta cualidad de AWS hace que la estimación de la puntuación que hacemos en función a los sentimientos que obtenemos tras el análisis sea más cercana a la media que hacemos de las puntuaciones que dejan los autores de los comentarios en Google Maps.

Abstract

Today we are in the information age, data has become a very valuable asset. Therefore, more and more companies want to take advantage of this asset. Resulting in large investments in the field of Machine Learning, both private and public.

Due to the growing interest in the field and the large investments in recent years, products focused on Machine Learning have emerged. These make available to companies and individuals not specialized in this technology tools that allow them to exploit the data they have.

This work focuses on studying the main options that currently exist related to Machine Learning. We focus on those that allow us to analyse texts, specifically, Natural Language Processing (NLP). During the market study, the options offered by the three main companies in the cloud computing sector are analysed: Amazon Web Services, Microsoft Azure and Google Cloud. We study the options offered by the three companies mentioned, choosing the one that best meets our needs at the lowest price.

Once the market study has been carried out, we have created an application to put into practice what these products offer us and to be able to compare the options of Amazon Web Services and that of Microsoft Azure.

The application consists of analysing the comments of different restaurants, classifying them based on two parameters: the language in which they are written and the feeling they convey. The source of the data is Google Maps. To perform the test, we have divided the scenario into four phases.

- In the initial phase we proceed to collect the information. For this we use the Google Maps API, specifically Place Details, using the telephone numbers of the premises to locate them within Maps.
- During the second phase we analyse the data we obtain from the first. For this phase we use the services of Amazon Web Services, specifically Comprehend and those of Microsoft Azure, specifically Text Analytics. In this phase we carry out two classifications. The first based on the language used in the comment and the second based on the feelings that said comments convey. We do it first with the AWS option and then with the Azure option. In the third phase we process the data to create statistics, for this we use the data we obtain from the previous phase. We count the comments left in each classification and create graphs that we will use in the next phase. The objective of this phase is to prepare the data to be able to visualize it in an agile way.
- In the fourth and final phase, we create an HTML document in which, from a template, we fill in the data and graphs obtained in the previous sections.

In this way, after finishing all the phases, the data has followed the following process:

Collection, analysis, processing, and visualization.

After finishing the commented phases, we compare the results we have obtained with both services.

In conclusion, we can say that both services have a similar behaviour, although AWS seems to "understand" the context of the sentence better than Azure, getting closer to what a human being could do. This quality of AWS makes the estimation of the score that we make based on the feelings that we obtain after the analysis is closer to the average that we make of the scores that the authors of the comments leave in Google Maps.

Índice

Agradecimientos viii

Resumen ix

Abstract x

Índice xi

Índice de Tablas xiii

Índice de Figuras xiv

1 Introducción 1
1.1 Introducción 1
1.2 Objetivos 2

2 Estado del Arte 5
2.1 Aprendizaje Automático 5
2.1.1 Usos del Aprendizaje Automático 6
2.2 Computación en la nube 7
2.3 Principales Actores de la computación en la nube. 8
2.3.1 Amazon Web Services (AWS) 8
2.3.2 Azure 9
2.3.3 Google Cloud 10
2.3.4 Valoración de los servicios estudiados 11
2.4 Google Maps 12

3 Diseño de la solución 14
3.1 Servicios que emplearemos 14
3.2 Descripción del escenario 14
3.3 Conclusiones 16

4 Implementación del escenario 17
4.1 Herramienta de AWS 17
4.1.1 Boto3 17
4.1.2 AWS CLI 18
4.2 Servicios de Google Maps. 18
4.3 Azure 21
4.4 Fase inicial 22
4.5 Fase de recolección de datos. 23
4.6 Fase de análisis 24
4.6.1 Análisis empleando AWS Comprehend 24
4.6.2 Análisis empleando Azure 28
4.7 Fase de tratamiento de los datos 30
4.8 Fase de visionado de datos. 30
4.9 Conclusión 32

5 Pruebas y funcionamiento 33
5.1 Diferencias entre los análisis de AWS y los de Azure 37
5.2 Coste de las pruebas 42

5.3 Conclusión *43*

6 Conclusiones **44**

6.1 Líneas Futuras *45*

Referencias **46**

Apéndice **49**

ÍNDICE DE TABLAS

Tabla 1.Productos de AWS. 8
Tabla 2.Productos de Azure I. 9
Tabla 3.Productos de Azure II. 10
Tabla 4. Productos de Google Cloud. 10
Tabla 5. Comparativa 11
Tabla 6. Valores para la puntuación media de los locales. 27

ÍNDICE DE FIGURAS

Ilustración 1. Datos de Google Maps Platform. 2
Ilustración 2. Implementación software del escenario. 3
Ilustración 3. Esquema de escenario. 14
Ilustración 4. Esquema de escenario con AWS. 15
Ilustración 5. Esquema de escenario con Azure. 15
Ilustración 6. Comando instalación Boto3. 17
Ilustración 7. Administrador de recursos AWS 17
Ilustración 8. Estructura comandos AWS CLI. 18
Ilustración 9. Configuración con aws configure. 18
Ilustración 10. Consola de Google Cloud. 19
Ilustración 11. Creando Clave de API. 19
Ilustración 12.Editar clave de API. 19
Ilustración 13. Seleccionando API a Restringir. 20
Ilustración 14. Ejemplo URL Petición *Place Details*. 20
Ilustración 15. Formato URL para solicitar el "Place ID" de Google Maps 20
Ilustración 16. Parámetros necesarios para solicitar el "Place ID". 20
Ilustración 17. Formato de la URL para solicitar los cometarios. 21
Ilustración 18. Parámetros necesarios para solicitar los comentarios. 21
Ilustración 19. Instalación librería Text Analytics. 21
Ilustración 20. Menú "Claves y conexión". 22
Ilustración 21. Llamada a `google`. 23
Ilustración 22. Llamada a `visionado_html`. 23
Ilustración 23. URL para solicitar el "Place ID". 23
Ilustración 24. URL para solicitar los comentarios de usuario. 24
Ilustración 25. Obtener los comentarios 24
Ilustración 26. Llamada a `analisisAWS` y `analisisAzure` 24
Ilustración 27.Conectar con AWS Comprehend. 24
Ilustración 28. Llamada método detect_dominant_Lenguage 25
Ilustración 29. Llamada método `detect_sentiment`. 25
Ilustración 30. Proceso de escritura en "OpinionAWS.txt". 26
Ilustración 31. Cálculo de puntuación media. 27
Ilustración 32. Error al generar la gráfica. 28
Ilustración 33. Error no existen comentarios. 28
Ilustración 34. Autenticación en Azure I. 28
Ilustración 35. Autenticación en Azure II. 28

Ilustración 36. Análisis de idioma con Azure. 29
Ilustración 37. Análisis de sentimientos con Azure. 29
Ilustración 38. Proceso de escritura en OpinionAzure.txt. 29
Ilustración 39. Comprobar si la llamada es desde AWS o desde Azure. 30
Ilustración 40. Guardando gráfica análisis de idioma de AWS. 30
Ilustración 41. Guardando gráfica análisis de sentimientos de AWS. 30
Ilustración 42. Diccionario para plantilla HTML I. 31
Ilustración 43. Diccionario para plantilla HTML II. 31
Ilustración 44. Diccionario para plantilla HTML III. 31
Ilustración 45. Número de comentarios por sentimiento 33
Ilustración 46. Número de hablante por idioma 33
Ilustración 47. Fragmento I de la visualización final del análisis de la prueba I. 34
Ilustración 48. Fragmento II de la visualización final del análisis de la prueba I. 35
Ilustración 49. Fragmento III de la visualización final del análisis de la prueba I. 35
Ilustración 50. Fragmento I de la visualización final del análisis de la prueba II. 36
Ilustración 51.Fragmento II de la visualización final del análisis de la prueba II. 36
Ilustración 52. Fragmento III de la visualización final del análisis de la prueba II. 37
Ilustración 53. Scores AWS. 37
Ilustración 54. Scores Azure. 38
Ilustración 55. Scores AWS II. 38
Ilustración 56. Scores Azure II. 38
Ilustración 57. Scores AWS III. 39
Ilustración 58. Scores Azure III. 39
Ilustración 59. Scores AWS IV. 40
Ilustración 60. Scores Azure IV. 40
Ilustración 61. Scores AWS V. 41
Ilustración 62. Scores Azure V. 41
Ilustración 63. Scores AWS VI. 41
Ilustración 64. Scores Azure VI. 42
Ilustración 65. Resumen factura agosto 2021. 42
Ilustración 66. Coste de AWS mes de agosto y septiembre del 2021. 43

1 INTRODUCCIÓN

En este capítulo de introducción mostraremos los objetivos que nos hemos marcado en el proyecto *Análisis de opinión mediante servicios de procesamiento natural con Amazon Comprehend de AWS*. Realizamos una visión panorámica del proyecto, mostrando los objetivos que nos hemos marcado e introduciendo al lector en el tema a tratar.

En el segundo capítulo, mostramos las opciones disponibles actualmente. Realizamos un estudio detallado de los principales actores que ofertan servicios de procesamiento de lenguaje natural basado en servicios en la nube. Además, introducimos conceptos y tecnologías necesarias para comprender el funcionamiento de los servicios en la nube y del *machine learning*. Una vez finalizado el estudio individual de cada uno de los actores, los comparamos para ver sus semejanzas y diferencias.

Tras esto, en los siguientes capítulos seleccionamos los servicios que mejor se adaptan a nuestro objetivo. Además, mostramos tanto el escenario que hemos planteado para las pruebas como la implementación que hemos realizado de este.

Por último, mostramos las pruebas y los resultados que hemos obtenido.

1.1 Introducción

Hoy día, prácticamente cualquier persona ha oído hablar sobre la inteligencia artificial o el *machine learning*. Nosotros mismos no hemos parado de escuchar estos términos durante el grado, aunque siempre desde lejos, sin profundizar demasiado. Durante la realización de este trabajo hemos tenido la oportunidad de entrar un poco más en este mundo. De empezar a vislumbrar cómo funciona realmente.

A pesar de que el campo de la inteligencia artificial, y por tanto del *machine learning*, comenzó a dar sus primeros pasos a mediados del siglo pasado. No fue hasta finales del siglo pasado y principios del actual cuando recibió un mayor impulso. Esto se debe a la unión de dos factores, el primero es que la capacidad de computación de los equipos ha aumentado de forma vertiginosa y el segundo ha sido que, desde la llegada de internet de forma generalizada, el volumen de datos que genera la humanidad diariamente ha aumentado exponencialmente. Para que nos hagamos una idea de la cantidad de datos que se generan cada año, Eric Schmidt, CEO de Google, afirmo en una conferencia en 2010 que hasta el año 2003 la humanidad había generado cinco exabytes de información a lo largo de toda su historia. En el año 2007 se generaron 281exabytes y en 2011 1800 exabytes [1].

Los modelos matemáticos y la ciencia que hay detrás del *machine learning* pueden ser muy complejos, requieren una gran formación y en muchos casos una potencia computacional que no está al alcance de la gran mayoría de usuarios o de empresas. Es por esto por lo que en los últimos tiempos han surgido productos destinados a poner al alcance de estos usuarios y empresas el *machine learning*. Estos productos están basados en la nube, por lo que para emplearlos no necesitamos equipos con procesadores o gráficas potentes. Además, se basan en el uso mediante APIs por lo que no requieren de conocimientos específico en el campo de la inteligencia artificial. Por estos motivos hemos realizado un acercamiento a la computación en la nube, centrándonos en aquellos servicios enfocados en el *machine learning*.

1.2 Objetivos

De esta forma nuestro objetivo ha sido estudiar los servicios que ofrecen las principales compañías que están el negocio de la computación en la nube. Centrándonos en aquellos servicios que nos permite emplear *machine learning*, mediante el uso de API que no nos exijan ser expertos en inteligencia artificial, si no que empleemos servicios que estén listos para usar. De esta forma podríamos resumir el alcance de nuestro proyecto en los siguientes puntos:

- Estudio del estado del arte con respecto a la computación en la nube, comparación de las principales opciones disponibles.
- Diseño de un escenario de pruebas para poner en práctica y comparar algunas de las tecnologías del estado del arte.
- Implementación software de dicho escenario.
- Pruebas de funcionamiento y muestra de los resultados de la implementación.
- Conclusiones.
- Líneas futuras.

Para la realización de las tareas expuestas, supondremos el caso de uso de una serie de clientes, dueños de locales de restauración, que quieren conocer lo que sus clientes opinan de los servicios que prestan en sus respectivos locales. Con esto, buscamos acercar esta tecnología al lector. El objetivo principal de esta prueba es determinar cómo los servicios de *machine learning* que vamos a presentar posteriormente, pueden ser aplicados de una forma sencilla en el mundo real. De modo que cualquier persona pueda emplear los datos que los clientes dejan, de forma voluntaria, en Google Maps para realizar un estudio de sus clientes. Para poder tomar medidas que le permitan mejorar el servicio que ofrece. La elección de Google Maps para la extracción de datos la basamos en dos puntos:

- El primero es centrándonos en el número de usuarios activos en la plataforma de Google Maps [8]. Según la propia Google, cuenta con más mil millones de usuarios activos por mes, un número muy elevado. Para ponernos en contexto es prácticamente la misma cifra que nos da Facebook sobre los usuarios activos en Instagram.

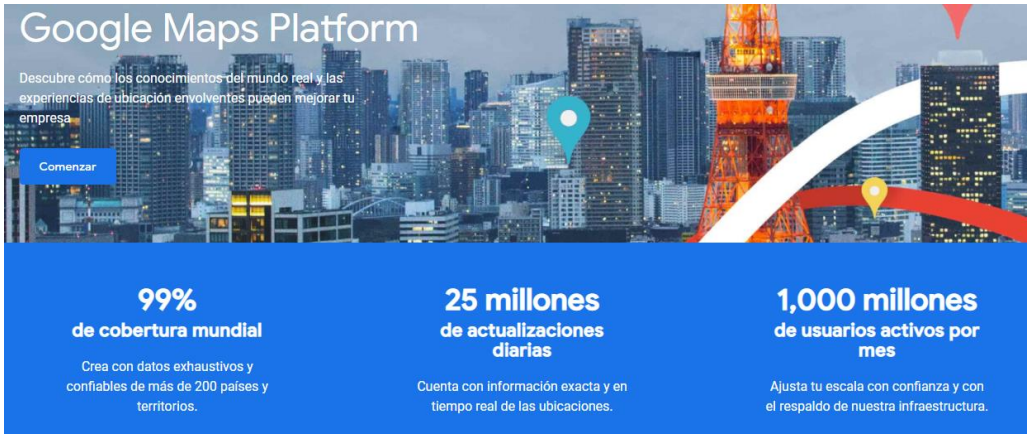


Ilustración 1. Datos de Google Maps Platform.

  Teniendo este número de usuario tan grande nos aseguramos una gran cantidad de datos.

- El segundo punto es que Google Maps tiene una API muy completa que nos permite integrar su uso en nuestra aplicación. Comentaremos la API en el siguiente capítulo.

Esto nos permite mostrar el potencial de esta tecnología, permitiéndonos a un coste muy bajo, como veremos en breves, mejorar el conocimiento que tienen los dueños de sus clientes. Hemos automatizado el proceso de

recopilación y análisis de los comentarios diseñando un programa en Python, tal y como mostramos a continuación.

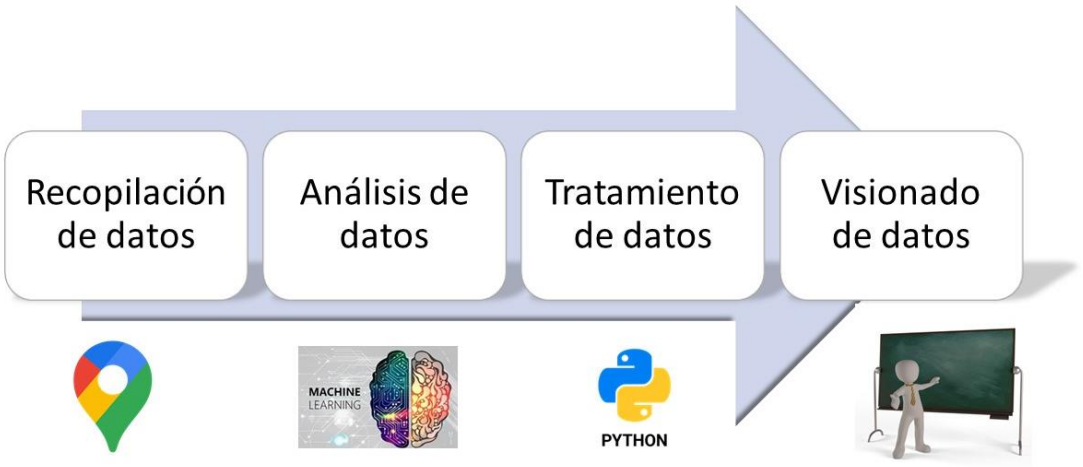


Ilustración 2. Implementación software del escenario.

En el esquema superior podemos observar cómo será la estructura de nuestro escenario.

En primer lugar, extraeremos los comentarios de los clientes de Google Maps, emplearemos los números de teléfono que tienen publicados en la plataforma para identificar a los restaurantes.

A continuación, realizaremos un análisis empleando *machine learning* para poder extraer información útil de los comentarios obtenidos en el paso anterior.

En tercer lugar, trataremos los datos para darles formato, los graficaremos para poder ver de forma ágil la información que hemos obtenido del análisis.

Por último, agruparemos las gráficas obtenidas y las mostraremos en un fichero HTML para que pueda ser visualizada en cualquier dispositivo compatible. De esta forma los datos habrán pasado por estas fases: recolección, análisis, procesado y visualización.

Con esto hemos concluido este primer capítulo de introducción. A continuación, pasaremos a explicar el estado del arte, donde analizaremos el panorama actual y explicaremos conceptos y tecnologías necesarias para nuestro trabajo.

2 ESTADO DEL ARTE

En este capítulo analizaremos el panorama actual. En primer lugar, realizamos un estudio sobre los términos y las tecnologías presentes en nuestro trabajo, introduciendo conceptos necesarios de manera que nos acerquemos a la tecnología implicada, mostramos también aplicaciones que tienen dicha tecnología. A continuación, estudiaremos las principales opciones disponibles actualmente de las principales empresas de la computación en la nube, enfocada al *machine learning*. Por último, realizamos un estudio sobre Google Maps, que será la fuente de nuestros datos. Resumiendo, al terminar este capítulo tendremos conocimiento de los siguientes temas:

- Aprendizaje Automático.
- Uso del aprendizaje automático en el análisis de sentimientos.
- Computación en la nube.
- Principales actores en la computación en la nube centrada en el *machine learning*.
- Google Maps, servicio de mapas.

2.1 Aprendizaje Automático

Antes de comenzar debemos tener claro que es el aprendizaje automático, en inglés *machine learning*.

A grandes rasgos el aprendizaje automático [1] es una rama de la inteligencia artificial que tiene como objetivo dotar a los ordenadores de la capacidad de identificar patrones en grandes volúmenes de datos. Este aprendizaje permite a los ordenadores realizar tareas específicas de forma autónoma, es decir, sin haber sido programados específicamente para dicha tarea. Les permite aprender.

Con el aprendizaje automático buscamos que introduciéndole a la máquina una serie de datos esta sea capaz de procesarlos y extraer información útil para nosotros. Claro está que esto no es algo inmediato, necesitamos entrenar a la máquina.

Llamamos **entrenamiento** [1] al proceso en el que se detectan los patrones de un conjunto de datos. Esto es el alma del *machine learning*. Una vez que hemos identificado los patrones podremos emplearlos para hacer predicciones con nuevos datos. Para entender esto un poco mejor vamos a hablar de las diferencias entre los sistemas supervisados y no supervisados. De entrada, puede parecer que los primeros se refieren a sistemas que están supervisados por humanos y los segundos no. La realidad es que tiene más que ver con que haremos con los datos.

Uno de los usos más extendido que se le dan al **aprendizaje supervisado** es el de predecir. Esto se consigue entrenando a la máquina. El entrenamiento lo realizamos con un conjunto de datos conocidos. Estos datos serían los que se conocen como datos de entrenamiento, estarían etiquetados por humanos y se suponen correctos. La máquina sería entrenada con este conjunto de datos para que extraiga patrones de estos. Una vez extraídos los patrones podríamos decir que la máquina ha aprendido. El objetivo de todo esto es que la máquina pueda reconocer estos patrones para clasificar ella misma los datos. Cabe destacar que es habitual que se reserven una cantidad de datos de disponibles, con el objetivo de comprobar si el entrenamiento ha ido bien o no. Es decir, para saber el grado de fiabilidad que nos ofrece. Nombramos los algoritmos más importantes actualmente:

- Regresión Lineal.
- Regresión Logística.
- Clasificación de Naïve Bayes.
- SVMs (Support Vector Machines).

Un ejemplo de esto podría ser la clasificación de correos electrónicos en función de si es o no spam. Para entrenar a la máquina emplearíamos un conjunto de datos, correos ya clasificados por humanos. Se los pasamos a la máquina y esta extraería patrones. Algunos patrones identificativos podrían ser la dirección de correo remitente de un correo mercado como spam o la dirección IP de la que proviene, entre otros.

El **aprendizaje no supervisado** en cambio usa datos históricos que no están etiquetados, de los que no disponemos de un conocimiento previo. El objetivo es estudiarlos para encontrar alguna forma de organizarlo, buscar patrones en ellos de forma que podamos extraer información útil de datos que antes no nos aportaban tanto valor. Algunos de los algoritmos más importantes que se emplean actualmente son:

- K-means.
- PCA (principals component analysis).
- Análisis de Componentes Independientes.

2.1.1 Usos del Aprendizaje Automático

Sus aplicaciones son prácticamente ilimitadas, podemos aplicarlo en todo lo que podamos imaginar.

Lo primero que se nos viene a la cabeza son las aplicaciones que le dan las grandes tecnológicas, empleando aprendizaje automático en los motores de búsqueda. También vemos como en sectores como el de la automoción se emplean técnicas de reconocimiento visual para, por ejemplo, permitir conducir de forma autónoma a un coche.

Sin embargo, existen sectores sobre los que su aplicación puede resultar clave, y no se nos vienen a la mente tan rápido. Como en el campo de la medicina, donde cada vez se emplean más técnicas basadas en *machine learning* para decidir si es necesario o no una operación, o para detectar diferentes enfermedades en la piel mediante reconocimiento de imágenes.

Podríamos seguir poniendo ejemplos y campos donde es posible aplicar aprendizaje automático hasta donde alcanzara nuestra imaginación.

En nuestro caso la aplicación será el análisis de las opiniones de usuario. Esto lo vamos a estudiar más detenidamente a continuación ya que existen términos que necesitan de una explicación para poder entenderlos.

2.1.1.1 Uso de Aprendizaje Automático para el análisis de opiniones

Las personas usamos lo que se denomina **lenguaje natural** (LN) para comunicarnos entre nosotros. Este lenguaje, dentro de la informática, sería catalogado como información no estructurada. Esto quiere decir que requiere de un procesamiento para poder sacarle partido. Este procesamiento es conocido como **procesamiento de lenguaje natural** (en inglés, NLP) y es necesario para poder tener la información estructurada.

El **análisis de sentimientos** (AS) es un campo dentro del aprendizaje automático que está cobrando importancia en los últimos años. Esto es debido a que cada día se genera mucha información subjetiva, creada por usuarios cuando dan su opinión sobre productos o servicios adquiridos. Esta información es muy importante para las empresas ya que es una fuente de mucho valor, les permite saber qué opinan sus clientes.

El gran volumen de datos que se generan actualmente hace necesario automatizar la tarea de revisar los comentarios. Es por esto por lo que surge el AS empleando aprendizaje automático, históricamente se han empleado múltiples técnicas para afrontar esta tarea. Clásicamente se han empleado técnicas de aprendizaje automático basado en algoritmos de aprendizaje supervisado como Naive Bayes, máquina de soporte de vectores o reglas de árboles de decisión, entre otros como se nos resume en [3].

Como vemos no existe un único enfoque a la hora de realizar este AS, por lo que siguen surgiendo artículos atacando este tema desde diversos puntos y con diferentes técnicas.

2.2 Computación en la nube

Antes de comenzar a estudiar las principales empresas presentes en los llamados servicios o la computación en la nube es necesario introducir qué es la computación en la nube.

La computación en la nube, o *cloud computing* en inglés [2] es un término que representa un nuevo modelo de la informática. La nube puede ser infraestructura o software, puede ser una aplicación a la que se accede a través del escritorio que se ejecuta tras su descarga, o bien un servidor al que se invoca cuando se necesita. Los servicios de la nube han de ser distribuidos, empresas diferentes compartirán los mismos recursos. Una de las principales ventajas que nos ofrece la computación en la nube es que pagaremos por lo que utilicemos y durante el tiempo que lo usemos. De forma que nos permite evitar grandes inversiones en licencias, equipos, energía y demás elementos que serían necesarios en la informática tradicional.

El *cloud computing* es uno de los campos que mayor crecimiento ha experimentado en los últimos años. Esto se debe a que permite que usuarios y empresas accedan a servicios y tecnología que hasta hace poco tiempo solo podían acceder las grandes empresas, por las grandes inversiones que requerían en la informática tradicional. A pesar de que nos pueda parecer que es un campo que acaba de surgir, fue a mediados del pasado siglo cuando se comenzó a hablar de la computación distribuida, de la mano de nombres como John McCarthy y Joseph Carl Robnett Licklider [6]. No obstante, no fue hasta la década de los noventa, cuando Internet ya contaba con un ancho de banda suficiente, cuando experimento un mayor crecimiento.

Las principales características de la computación en la nube son las siguientes [4]:

- Una de las más importante es que no requerimos de un equipo con las últimas tecnologías. Nos basta con que tenga conexión a Internet.
- Autoservicio a la carta. Es el consumidor el que decide qué servicios y cuando consumirlos. La empresa ofrecerá ese servicio de forma automática, sin intervención de un operador humano.
- Reservas de recursos en común. El proveedor oferta sus servicios a múltiples clientes, los cuales comparten dichos recursos que son asignados de forma dinámica.
- Rapidez y elasticidad. Para el usuario los recursos pueden aparecer como ilimitados y es el proveedor el que realiza el reparto y el redimensionamiento de forma automática.
- Servicio supervisado. Los sistemas en la nube controlan los recursos disponibles de forma automática empleando medidas de control en diferentes puntos. El uso de los recursos es medible y controlable, lo que aporta transparencia a ambas partes.
- Es auto reparable. En caso de fallo el último respaldo se convierte automáticamente en la copia primaria. De forma totalmente transparente para el usuario.
- Es escalable. Todo el sistema es predecible y eficiente, por lo que escalarlo es inmediato para el proveedor.
- Virtualización. El usuario es libre de usar el sistema operativo que desee, al emplear aplicaciones en la nube tiene la seguridad de que conservara las mismas características independientemente de la plataforma.
- Posee un alto nivel de seguridad. Debido a la forma en la que esta creada, diferentes clientes pueden emplear los mismos recursos sin que sus datos se vean comprometidos. El proveedor se encarga de cifrar los datos.
- Disponibilidad de la información. No es necesario que el usuario almacene información de su trabajo en su propio dispositivo. Esta estará disponible a través de internet por lo que podrá acceder desde cualquier dispositivo y en cualquier lugar.

Todas estas características son las que han hecho que la computación en la nube crezca con tanta fuerza estas últimas décadas.

2.3 Principales Actores de la computación en la nube.

Una vez definido que es la computación en la nube, nos encontramos con tres grandes proveedores, Google, Amazon, y Microsoft. Todas ellas ofertan multitud de productos, tanto software como hardware para dar solución a problemas de cualquier índole. Incluyendo al aprendizaje automático.

Existen opciones que permiten alojar servicios de *machine learning* complejos, brindando la potencia necesaria para entrenar una máquina. Proporcionando, además, ayuda en su diseño. Por otro lado, también tienen servicios accesibles mediante API permitiendo ejecutarlo en aplicaciones que lo requieran. Empleando servicios ya entrenados. Nos centraremos en estos últimos.

A continuación, mostraremos un resumen, el cual nos permitirá comparar entre los tres proveedores para elegir el que se adapte mejor a nuestras necesidades. Si se desea más información aconsejamos consultar la bibliografía con el fin de ir a la web de los proveedores para ver la documentación completa.

Tenemos que destacar que todas las empresas ofrecen alguna fórmula u opción para probar en su totalidad, o de manera parcial, los servicios de forma gratuita. Al menos hasta un determinado límite que varía en función del producto concreto, con el objetivo de atraer a nuevos usuarios. Nosotros nos centraremos en aquellos que se puedan usar de forma gratuita e indicaremos, siempre que sea posible, cuando comenzará a ser de pago.

2.3.1 Amazon Web Services (AWS)

Como se deduce de su nombre, este es el servicio que ofrece Amazon.

Para ver las opciones disponibles basta con que nos dirijamos a su página web [3]. Una vez en ella vemos que existen productos que tienen los que Amazon denomina “capa gratuita” esto quiere decir que mínimo cuentan con una prueba gratuita de doce meses de forma que nos permitirá familiarizarnos con ellos, aunque pueden tener algunas limitaciones extras. Los agrupamos en la siguiente tabla. [4]

Tabla 1.Productos de AWS.

Nombre del producto	**Descripción**	**Límite**
Amazon Comprehend	Procesamiento de lenguaje natural (NLP) que emplea machine learning para descubrir información en datos no estructurados.	50 000 unidades de texto (5 millones de caracteres) por cada API por mes. 5 trabajos de modelado de temas de hasta 1 MB cada uno por mes durante los primeros 12 meses.
Amazon Lex	Cree chatbots conversaciones de voz y texto.	10000 solicitudes de texto por mes, 5000 solicitudes de voz por mes.
Amazon Polly	Transforma texto en habla verosímil.	5 millones de caracteres por mes.
Amazon Rekognition	Servicio de reconocimiento de imágenes basado en el aprendizaje profundo.	5000 imágenes por mes, almacene hasta 1000 metadatos de rostros por mes.
Amazon Trasncribe	Agregue capacidad de conversión de habla a texto a sus aplicaciones con reconocimiento automático del habla.	60 minutos por mes

Amazon Translate	Servicio de traducción automática neuronal ágil, económico y de alta calidad.	2 millones de caracteres por mes.

2.3.2 Azure

Este sería el servicio ofertado por Microsoft.

Primero, queremos destacar que Azure nos da el primer mes gratuito, limitado a un crédito de 170 €, tras agotar el crédito nos avisará antes de cobrarnos para que podamos actualizar la cuenta. Si no la actualizamos finalizará los procesos que podamos tener activos hasta que se autorice, entonces pasaría a cobrarnos por uso. Hay que destacar que estos 170 € euros de crédito solo pueden gastarse en aplicaciones propias de Azure y no en servicios de terceros que puedan estar relacionados con Azure. También posee una subscrición para estudiantes universitarios [8] de forma que permite probar muchas de sus tecnologías de forma gratuita durante un año, sujeto a un crédito de 100USD, pero sin necesidad de introducir tarjeta de crédito ni ningún otro medio de pago.

A continuación, recogemos los servicios disponibles con Azure relacionados con IA y Machine Learning. [5]

Tabla 2.Productos de Azure I.

Nombre del producto	**Descripción**	**Límite**
Azure Bot Service	Ofrece un servicio de desarrollo y conectividad escalable e integrado para bots inteligentes que pueden usarse para llegar a los clientes a través de varios canales.	Gratis para siempre en los canales estándar. Que son los canales propios de Microsoft o los de código abierto como el de Facebook. Para canales premium, que son los que permiten al bot comunicarse con confianza con los usuarios en sitios web o aplicaciones propias, será gratis hasta 10000 mensajes por mes
Azure Cognitive Search	Servicio que aplica machine learning en la búsqueda en la nube, aplica técnicas de aprendizaje profundo para ayudarlo a identificar y explorar contenido relevante.	Tienes muchas variables dentro de lo gratis.
Microsoft Genomics	Es un servicio escalable y seguro para el análisis secundario de genomas. Empieza a partir de lecturas de datos sin formato y produce lecturas alineadas y llamadas a variables.	No es gratuito nunca, hasta 10 Giga bases tiene precio fijo, luego tiene incrementos.
Azure Machine Learnig	Servicio de aprendizaje automático de nivel empresarial para crear e implementar modelos con más rapidez	No es gratis, su precio aumenta en función de los recursos que queramos emplear para el entrenamiento.
Machine Learning Studio	Es un conjunto de ofertas diseñado para permitir a los clientes crear, implementar y compartir con	Tiene una versión que es gratis para siempre.

	facilidad soluciones de análisis avanzado en la nube.	

Aparte de los servicios recogidos en la tabla anterior tenemos los denominados *Cognitive Services*. Donde podemos encontrar diferentes servicios que se agrupan en tres bloques: visión, lenguaje y búsqueda. Para acotar estudiaremos los relacionados con el lenguaje.

Tabla 3.Productos de Azure II.

Nombre del producto	**Descripción**	**Límite**
Text Analytics API	Es un servicio basado en la nube que ofrece procesado de texto sin formato. Incluye tres funciones principales: detección de idioma, detector de sentimiento y extraer frases claves.	Sin límite en la instancia "Gratis – Web/Container". En esta instancia quedaría excluida la detección de sentimientos. 5000 peticiones por mes para estudiantes universitarios, incluido la detección de sentimientos.
Language Understanding (LUIS)	Ofrece una forma rápida y eficiente de incorporar comprensión lingüística a las aplicaciones. Puede utilizar modelos precompilados o, si lo necesita, LUIS nos guiará para crear uno.	Está limitado a 5 transacciones por segundo y solo se podrán efectuar 10.000 transacciones por mes gratis.
Text Anlytics	Extraer conocimiento de texto no estructurado mediante procesamiento de lenguaje natural.	Gratis hasta 5000 transacciones por mes.
Mocrosoft Traslator Text API	Servicio de traducción automática compatible con varios idiomas.	2 millones de caracteres de cualquier combinación de traducción estándar y entrenamiento personalizado gratis por mes.

2.3.3 Google Cloud

Google nos ofrece probar todos sus servicios con un crédito de 300 $ durante un año o hasta que lo agotemos. Una vez finalizado el periodo de prueba el sistema suspenderá los procesos que tengamos activos mientras no actualicemos la cuenta, sin cobrar nada hasta entonces. Tendremos 30 días antes de que se elimine nuestra información. También tiene varios servicios que se pueden usar de forma gratuita permanentemente, hasta unos límites, diferentes para cada producto

A continuación, recogemos los productos que son gratis para siempre, junto con su descripción y su límite. [6]

Tabla 4. Productos de Google Cloud.

Nombre del producto	**Descripción**	**Límite**
Speech-to-Text	Se le envía un archivo de audio o lo transcribe	Gratis hasta 60 minutos de audio.

Cloud Video Intelligence API	Permite anotar videos almacenados de forma local o en la nube con información contextual. También tiene funciones para transcribir y detectar características del video.	Gratis los primeros 1000 minutos de mes.
Dialogflow	Es un sistema que ayuda en la comunicación entre un usuario de una aplicación y la aplicación. Traduce expresiones del lenguaje natural, tanto escritas como por audios, a comandos que la aplicación pueda entender.	Tiene una versión estándar que es gratis siempre, pero con limitaciones en el número de solicitudes. Estas varían en función de si son de audio o escritas.
Cloud Translation	Conjunto de funciones enfocada a la traducción de texto.	Gratis hasta 10 $ al mes.
Cloud Vision	Conjunto de funciones para el análisis de imágenes.	Se cobra por imagen enviada, las primeras 1000 imágenes de cada mes son gratis.
Natural Language	Proporciona un conjunto de funciones para analizar texto no estructurado.	Se paga solo por las funciones que se usen. Los documentos que se envían se cuentan en unidades, cada 1000 caracteres conforman una unidad. De 0 a 5000 unidades al mes es gratis. Excepto la función de clasificación de contenido que es hasta 30000 unidades por mes.

2.3.4 Valoración de los servicios estudiados

Ahora vamos a comparar los productos de las tres empresas para poder identificar sus semejanzas y sus diferencias. Puesto que nuestro trabajo se centra en el procesamiento de lenguaje natural (en inglés, NLP), hacemos foco en los servicios destinados a este fin.

Tabla 5. Comparativa

AMAZONE WEB SERVICES (AWS)	Amazon Comprehend	50000 unidades de texto (5millones de caracteres) por mes en los primeros doce meses
MICROSOFT AZURE	Text Analytics	5000 transacciones gratis por mes
GOOGLE CLOUD	Natural Language	Gratis hasta 5000/mes

Como podemos ver en la tabla superior, los tres proveedores poseen servicios NLP y los tres nos permiten probar sus servicios con unas limitaciones prácticamente idénticas. Por tanto, esto no será un punto decisivo para decantarnos por uno u otro servicio. En muchas ocasiones encontraremos que la elección entre los proveedores dependerá del contexto del proyecto y de cómo se adapten los servicios en la nube que buscamos al resto de herramientas que estemos empleando ya. En el siguiente capítulo mostraremos y justificaremos nuestra elección.

2.4 Google Maps

En esta sección vamos a explicar la plataforma de mapas de Google, Google Maps, concretamente estudiaremos la API "**The Places API**" que es un servicio que devuelve información sobre lugares usando peticiones HTTP.

Las solicitudes de lugares disponibles son [8]:

- ***Place Search***: devuelve una lista con lugares basados en las ubicaciones del usuario o en una cadena de caracteres.
- ***Place Details***: devuelve información más detallada sobre un lugar concreto, incluidas *reviews* de usuarios.
- ***Place Photos***: Nos da acceso a los millones de fotos, relacionadas con el lugar especificado. Almacenadas en la base de datos de Google Place.
- ***Place Autocomplete***: relleno automático de nombre y/o direcciones de lugares a medida que los usuarios lo escriben.
- ***Query Autocomplete***: proporciona un servicio de predicción para las búsquedas geográficas basadas en texto, devolviendo consultas sugeridas a medida que los usuarios escriben.

Todos estos servicios son accesibles con peticiones HTTP, las respuestas se producen con formato JSON o XML. Todas las peticiones al servicio Places tienen que usar el protocolo HTTPS e incluir una API key.

La API emplea lo que llaman "***place ID***" para identificar de forma inequívoca a cualquier lugar o emplazamiento.

De las funciones que hemos expuesto arriba, las más interesantes para nosotros serán la "***Place Details***" y la "Place photos". Por lo que las comentaremos con más detalle:

- ***Place Photos***: Es una API únicamente de lectura que nos permitirá agregar contenido fotográfico de alta calidad a nuestra aplicación. El servicio nos brindará acceso a la base de dados fotográfica de Place Photos. Todas las peticiones a Place Photo deben incluir una "***photoreference***", devuelta en la respuesta de una solicitud de búsqueda de lugar, búsqueda cercana, búsqueda de texto o detalles de lugar. La respuesta a estas solicitudes contendrá un campo "photo[]" si el lugar tiene contenido fotográfico relacionado. El número de fotos devuelta dependerá de la petición. Cada elemento *photo* contendrá los siguientes elementos:
    - ***photo_reference***: una cadena de texto empleada para identificar a la foto cuando realizamos una petición de foto.
    - ***height***: la máxima altura de la imagen.
    - ***width***: el máximo ancho de la imagen
    - ***html attributions []***: contiene las atribuciones requeridas. Es un campo que siempre existirá, aunque puede estar vacío.

    Si la foto recibida no trae el campo *html_attributions []* vacío, deberá mostrarse dicho campo si mostramos la foto en nuestra aplicación.

    La petición para emplear Place Photo Requests:

    - "***https://maps.googleapis.com/maps/api/place/photo?parameters***", los parámetros estarán separados entre ellos por "&" y son los siguientes:
    - ***key***: Nuestra "API key". Es la llave que identifica a nuestra aplicación para poder controlar nuestras cuotas.
    - ***photoreference***: Es una cadena de texto que identifica de forma exclusiva una foto. Esta referencia se devuelve en las solicitudes de búsqueda de lugar o detalles de lugar.
    - ***maxheight or maxwidth***: Especifican el máximo del alto o del ancho, en píxeles, de la imagen devuelta. Si la imagen es menor que los valores especificados, se devolverá la imagen en su tamaño original. Si la imagen es más grande se escalará para que coincida con la menor de las

dos dimensiones, se respetará la relación de aspecto original. Este parámetro acepta un valor entero entre 1 y 1600.

- ***Place Details***: Una vez que tengamos el "**place_id**" de un lugar podemos solicitar más detalles sobre el lugar en cuestión. Estos detalles son los que solicitaremos mediante la petición de Place Details. Algunos campos que podemos pedir son la dirección completa, el número de teléfono, su calificación de usuario y, lo que más nos interesa a nosotros, los comentarios de usuario o reviews. Para realizar la consulta se empleará el mismo método que para la comentada antes, los parámetros en este caso son los siguientes:
    - ***key***: Al igual que antes, es la "API key" de nuestra aplicación.
    - ***places_id***: Es un identificador de texto que identifica de forma inequívoca al lugar, es devuelto por *Place Search*.
    - ***fields***: especifica que tipos de datos estamos solicitando, los campos de *fields* irán separando por comas. Los campos se pueden dividir en tres: básicos, de contacto y atmósfera. Los campos de básico se facturan a la tasa base y no incurren en cargos adicionales. Los otros dos se facturarán a una tasa más alta. Los detalles de precio se especifican en la hoja de precios.
        - ***Basic***: Incluye las siguientes categorías: addres_component, adr_addres, business_status, formatted_address, geometry, icon, name, permanently_closed, photo, place_id, plus_code, type, url, utc_offset, vivinity.
        - ***Contact***: Incluye las siguientes categorías: formatted_phone_number, international_phone_number, opening_hours, website.
        - ***Atmosphere***: Incluye las siguientes categorías: Price_level, rating, review, user_ratings_total. La más interesante para nosotros es review, que nos devuelve un objeto JSON con información de las review.

    Los tres primeros parámetros que hemos descrito son obligatorios, en cambio, *fields* es opcional. Existen más campos opcionales, pero no son de interés para nuestro estudio.

Ya hemos comentado los servicios que nos ofrecen, profundizando en los que vamos a usar. Ahora vamos a exponer que nos va a costar usar estos servicios.

2.4.1.1 Precio de Google Maps

Lo primero que debemos tener en cuenta, es que Google nos proporciona 200 USD al mes de crédito. Si no nos pasamos de esta cantidad podremos usar los servicios comentados arriba sin coste. También nos da la opción de fijar límites diarios para evitar sorpresas respecto a lo que gastemos.

En un principio, para nuestras pruebas este crédito será suficiente. Para darle mayor alcance a nuestro estudio realizaremos un resumen de los precios. Debemos destacar que los precios varían según el intervalo de consultas en el que nos encontremos. A continuación, mostraremos el precio de los dos servicios descritos antes.

- Para el servicio de Places Photos el precio será de 7 $ por cada mil solicitudes.
- Para el servicio de Place Details el precio variará en función del grupo en el que se encuentre el dato que queremos solicitar. El precio base será de 17 $ para los básicos incluidos, se incrementará 3 $ si añadimos los de contacto y 5 $ si añadimos los atmosféricos. [9]

En conclusión, como hemos visto, el campo del *machine learning* va cobrando cada vez más peso. Motivo por el que cada vez surgen más productos en dicho campo, creando una competencia entre las empresas implicadas. Como se ha podido ver, las tres opciones estudiadas tienen unos servicios similares, a precios similares, lo que provoca que las tres opciones terminen siendo válidas simultáneamente en muchos casos.

En el siguiente capítulo mostraremos y justificaremos nuestra elección. Tras esto, estudiaremos las diversas herramientas que poseen los diferentes servicios que vamos a emplear en nuestra solución. Además, procederemos a explicar cómo será nuestra solución a alto nivel.

3 DISEÑO DE LA SOLUCIÓN

En este capítulo comenzaremos por seleccionar que servicios, de los presentado en el capítulo anterior, vamos a emplear. Eligiendo los que mejor se adapten a nuestra aplicación. Tras esto comenzaremos a describir a alto nivel la estructura de la solución que planteamos. También mostraremos como vamos a realizar el análisis de sentimientos sobre los comentarios que extraemos de Google Maps. Para finalizar mostraremos como podemos realizar el mismo análisis con la opción de Microsoft, Azure. Por tanto, al finalizar el capítulo se habrá tratado lo siguiente:

- Elección y justificación de nuestro proveedor de servicios en la nube.
- Flujo de trabajo de nuestra aplicación.

3.1 Servicios que emplearemos

Como hemos comentado ya, las tres opciones son similares, y ofrecen los servicios que buscamos para nuestra aplicación.

En nuestro caso emplearemos los servicios de AWS y de Microsoft Azure para nuestros análisis. Lo haremos por dos motivos:

1. A pesar de que los tres nos ofrecen un límite similar, AWS es quién más cuota de mercado posee [7] en productos de computación en la nube, con un 32% frente al 20% de Microsoft Azure y el 9% de Google Cloud. Por tanto, emplearemos las dos compañías con mayor cuota de mercado.
2. Ya vamos a emplear los servicios de Google para la extracción de los datos mediante una de las API de Google Maps. Por tanto, terminamos empleando las tres principales compañías en nuestro trabajo.

Ya que ambos proveedores nos ofrecen productos muy similares implementaremos nuestro escenario con ambos servicios de forma que podamos comparar y ver cómo se comportan ambas.

3.2 Descripción del escenario

Para proceder a explicar cómo es nuestro escenario, vamos a ayudarnos del siguiente esquema.

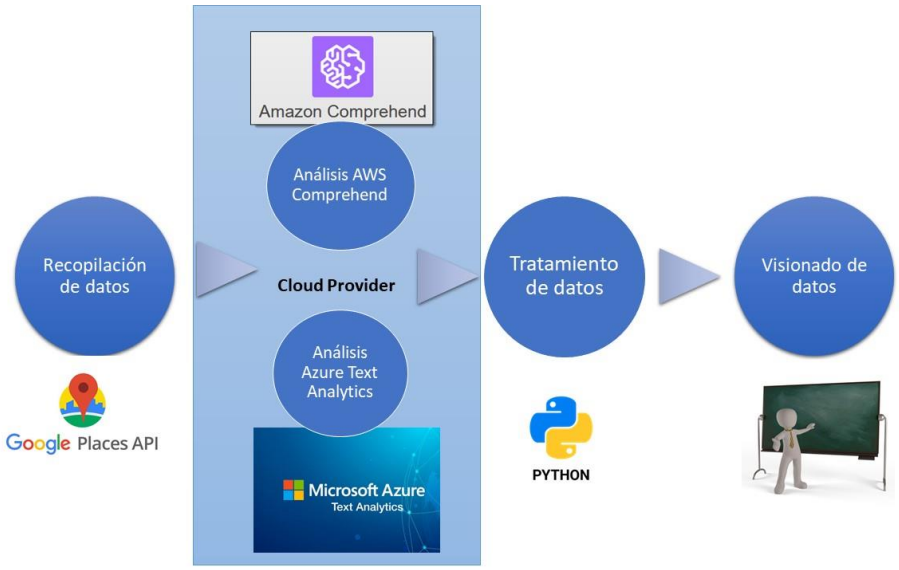


Ilustración 3. Esquema de escenario.

En el esquema superior podemos ver que APIs empleamos en nuestra solución. Podemos observar que las APIs correspondientes a los servicios de análisis están englobadas en un bloque llamado *Cloud Provider*, proveedor de nube en castellano. Este bloque simboliza que la fase de análisis se realiza empleando los servicios de ambas compañías, AWS y Azure, pero al realizar los mismos análisis, primero con una y luego con otra los podemos englobar en un único bloque. Realizamos el mismo análisis con ambos servicios para poder comparar los resultados al final. Con el fin de organizar la explicación, la hemos dividido en fases, que se suceden secuencialmente.

- La primera fase es la de recolección de la información. Aquí empleamos los servicios de la API *Place Details* de Google Maps. Usamos los números de teléfono de los restaurantes para localizarlos en Maps y poder extraer los comentarios de estos.
- La segunda fase es la de análisis. Una vez finalizada la recolección debemos de analizar los datos. Esto lo hacemos empleando los *Cloud Provider*, en nuestro caso, los servicios de AWS *Comprehend* y de Microsoft Azure *Text Analytics*. En esta fase llevamos a cabo dos análisis, el primero detecta el idioma en el que está escrito el comentario y el segundo analiza los sentimientos que caracteriza a este. Esto último nos clasifica el comentario en "positive", "negative", "mixed" o "neutral". Como podemos ver en el esquema realizamos los dos análisis con ambos servicios, los resultados de dichos análisis serán tratados en la siguiente fase. La siguiente fase distinguirá el origen de los datos, si provienen de AWS o de Azure, con el fin de poder mostrarlos en la última fase, *visionado de datos*, por separado. Para poder entender esto último mostramos como sería el escenario separando los *Cloud Provider*.

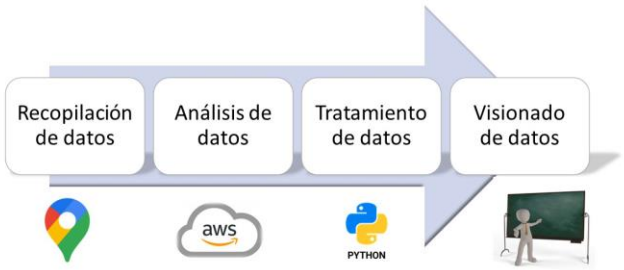


Ilustración 4. Esquema de escenario con AWS.

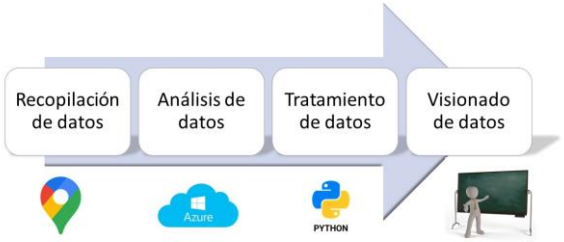


Ilustración 5. Esquema de escenario con Azure.

- En la tercera fase, tratamos los datos para poder crear estadísticas. Lo primero que hacemos en esta fase es contar el número de comentario que hemos analizado. A continuación, contamos cuantos comentarios hay en cada idioma contemplado. Para finalizar, numeramos los comentarios que hay con cada una de las etiquetas que surgen del segundo análisis. A cada uno de los sentimientos detectados le asignamos una puntuación, con el objetivo de obtener una valoración media del restaurante, algo similar a la clasificación por puntos o estrellas que encontramos en muchas webs. La puntuación que damos es la siguiente:
    - Positive=5.
    - Negative=-1

  - Mixed=3.5
  - Neutral=2.5

  Como vemos, al tener *Negative* una puntuación negativa, deberemos tener cuidado y truncar el resultado a cero en el caso de obtener una puntuación media negativa.

- Tras finalizar el análisis y el procesado de los datos pasamos a la visualización. En esta fase creamos un fichero HTML con el fin de mostrar las gráficas comparativas de los restaurantes analizados, permitiéndonos visualizar el contenido de forma directa y fácil.

Para que la comparativa tenga valor, ambas opciones deben analizar los mismos datos. Es por esto por lo que primero extraemos los comentarios de Google Maps y tras esto le pasamos las dos API de análisis (*Comprehend y Text Analytics*), una vez completado ambos análisis comparamos los resultados para ver cómo se han comportado las API. Explicamos este proceso más detalladamente en el próximo capítulo.

3.3 Conclusiones

Como hemos podido ver a lo largo de este capítulo, la aplicación de las tecnologías presentada en el capítulo anterior permite resolver problemas mediante *machine learning* empleando los servicios de proveedores a través de sus API. Esto permite acercar está tecnología a un mayor público, permitiendo crear soluciones más baratas y en menor tiempo. También hemos visto cómo podemos combinar los servicios de diferentes compañías en nuestra solución e, incluso, realizar una misma tarea con más de un servicio para compararlos posteriormente y ver qué servicio tiene un comportamiento más cercano al análisis que podría realizar una persona.

En el siguiente capítulo procedemos a explicar a más bajo nivel la estructura de nuestra solución, comentando los métodos creados para realizar cada una de las fases ya explicadas y como colaboran estos métodos entre sí.

4 IMPLEMENTACIÓN DEL ESCENARIO

En esta sección vamos a explicar con más detalle el funcionamiento de nuestro escenario. Introduciremos las herramientas necesarias para poder emplear las APIs en nuestro ordenador, que librerías necesitamos instalar y como debemos de proceder para crear nuestro proyecto en los distintos proveedores de servicios en la nube. Una vez finalizado esto, procederemos a explicar cómo hemos codificado las fases explicadas en el capítulo anterior.

4.1 Herramienta de AWS

AWS nos proporciona una serie de herramientas que nos permiten usar sus servicios en nuestro ordenador de forma sencilla e integrarlos en nuestra aplicación. A continuación, mostramos las que hemos usado para este proyecto.

4.1.1 Boto3

Para una primera definición vayamos a la que nos da Amazon en su página oficial [14]. Boto3 es el SDK, kit de desarrollo software, para Python que ofrece Amazon que facilita la integración de su aplicación, biblioteca o script de Python con los servicios de AWS, incluidos Amazon S3, Amazon EC2, Amazon DynamoDB y más.

Esto no es más que un conjunto de herramientas que nos sirve para facilitar la integración de los servicios de AWS con nuestra aplicación.

El uso de Boto3 es muy sencillo, para poder usarlo en nuestro equipo solo requerimos instalar la librería que tiene para Python mediante la siguiente sentencia:

```
pip install boto3
```

Ilustración 6. Comando instalación Boto3.

Tras esto ya tendremos instalada la última versión estable de la librería, cabe destacar que, según la web de AWS, Boto3 ha sido diseñado para ser compatible con las versiones 2.7+ y 3.4+ de Python.

Una vez instalado Boto3 necesitaremos pasarle nuestras credenciales, dichas credenciales se encuentran en el panel *Identity and Access Management* IAM de AWS. A dicho panel accedemos desde el buscador de la página de AWS, introduciendo las siglas “IAM”, como mostramos en la siguiente imagen.

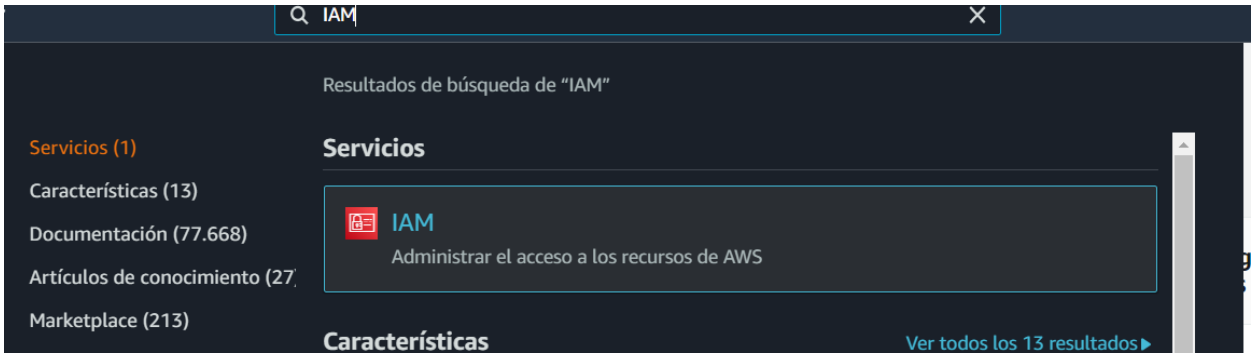


Ilustración 7. Administrador de recursos AWS

Desde aquí podemos crear y eliminar usuarios, asignándole diferentes permisos, lo que permitirá a dichos usuarios acceder a determinados recursos. Los usuarios tendrán unas claves y unas ID que nos permitirán usarlos en nuestro equipo. Las claves de los usuarios solo serán visibles durante su creación, una vez finalizada no se volverá a mostrar y deberemos crear una en caso de hacernos falta, por lo que conviene apuntarla.

Existen varios métodos para usar los usuarios IAM [15]:

- AWS Management Console
- AWS Command Line Tools
- AWS SDKs
- IAM HTTPS API

Nosotros empleamos *AWS Command Line Tools* concretamente la opción de *AWS Command Line Interface* (AWS CLI) que explicamos a continuación.

4.1.2 AWS CLI

Es la interfaz de línea de comandos de AWS, CLI por sus siglas en ingles. Según la documentación que nos ofrece Amazon [16] es una herramienta unificada para administrar los productos de AWS. Descargando esta herramienta podremos configurar y administrar todos los servicios de AWS desde la línea de comando de nuestro sistema, pudiendo automatizarlo mediante secuencia de comandos. Está disponible para Mac, Linux y Windows. En este último caso tiene versiones de 32 y de 64 bits. En nuestro caso hemos necesitado la versión de 64 bits para Windows. Tras instalar AWS CLI, podremos acceder a las funciones mediante una serie de comandos, estos comandos siempre comienzan con *aws* [19] tal y como podemos ver en la siguiente imagen:

```
aws [options] <command> <subcommand> [parameters]
```

Ilustración 8. Estructura comandos AWS CLI.

Dentro de las muchas funciones que posee CLI nosotros lo hemos usado para configurar nuestro usuario y nuestra región de trabajo de AWS en nuestro equipo, de esta forma podemos emplear los servicios en nuestra aplicación sin necesidad de introducir ninguna clave en nuestro archivo de Python. En la propia web de AWS nos muestra un ejemplo de cómo se hace como mostramos a continuación [17]:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: ENTER
```

Ilustración 9. Configuración con aws configure.

De esta forma ya tenemos los datos del usuario que creamos previamente, por lo que AWS nos permitirá usar dicho usuario en nuestro equipo y podrá registrar el uso que hacemos de los servicios para poder facturarlos.

4.2 Servicios de Google Maps.

La API que usamos de Google Maps está dentro de los servicios de Google Cloud, por lo que requiere la creación de una cuenta en dicha plataforma.

En este caso hacemos uso de la API mediante peticiones HTTPS, para que funcionen correctamente estas peticiones debemos de hacer, previamente, una serie de configuraciones en nuestra cuenta de Google Cloud.

Lo primero es crear un nuevo proyecto, esto se hace forma muy sencilla en la consola de Google Cloud, justo donde señalamos en la siguiente imagen:

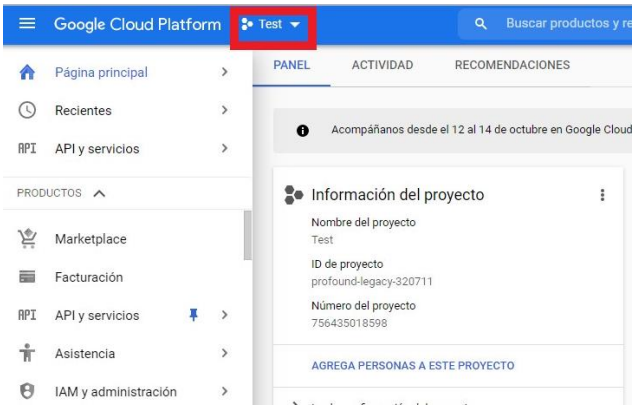


Ilustración 10. Consola de Google Cloud.

Clicando donde señamos en rojo y seleccionando "nuevo proyecto" podemos crear un nuevo proyecto simplemente introduciendo el nombre que deseemos. Una vez creado el nuevo proyecto buscamos en la biblioteca de API la *Place API* y le damos a habilitar.

Una vez hemos habilitado la API que queremos usar deberemos de crear una clave, *clave de API*, para poder acceder a ella desde nuestra aplicación. Esta clave es usada por Google Cloud para identificarnos como usuario, verificando la cuota y el acceso a la API, por tanto, es una clave única y privada y tendremos que incluirla en todas las peticiones que hagamos para poder usar la API. Para crear la clave deberemos de irnos a "API y servicios" y desde allí a "Credenciales". Aquí le damos a "crear credenciales", en el desplegable que surge seleccionamos la opción "Clave de API", tal y como mostramos en la siguiente imagen:

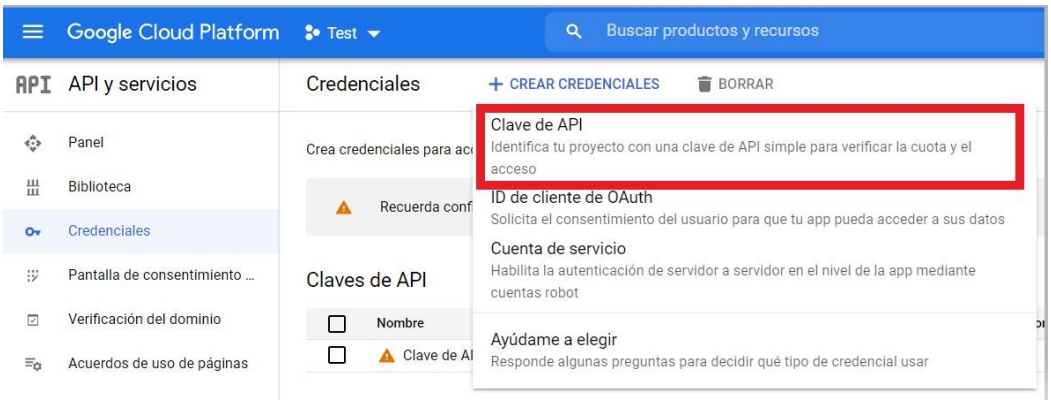


Ilustración 11. Creando Clave de API.

En nuestro caso empleamos la API de Google Maps por lo que limitaremos el uso de la clave a dicha API, para ello editaremos la Clave API que hemos creado, clicando en el lápiz que hemos señalado en rojo.

Ilustración 12.Editar clave de API.

A continuación, vamos a restringir los servicios que podemos emplear con la clave que hemos creado. El objetivo de limitar la clave es restringir los servicios que podemos usar con dicha clave, de forma que nos protegemos ante posibles errores. Para hacerlo, seleccionamos "Restringir clave" y buscamos la API que vamos a usar tal y como mostramos en la siguiente imagen.:

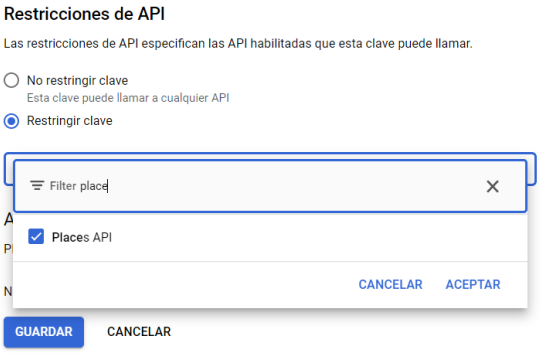


Ilustración 13. Seleccionando API a Restringir.

Una vez completado, podremos hacer uso, únicamente, de la API seleccionada en nuestra aplicación, para ello solo necesitaremos la clave que hemos generado.

Accederemos a los servicios de Google mediante peticiones HTTP, concretamente emplearemos el método GET. Para ello haremos uso de la estructura de URL que Google Cloud tiene para dicho fin, esta estructura puede variar en función del servicio que empleemos. Pondremos de ejemplo la que deberemos de emplear para usar *Place Details*, que seguirá la siguiente estructura [21]:

`https://maps.googleapis.com/maps/api/place/``details/output?parameters`

Ilustración 14. Ejemplo URL Petición *Place Details*.

Donde "output" puede ser "json" de forma que la salida tendrá notación *JavaScript Object Notation* (JSON, por sus siglas en inglés), o, "xml" para que la salida tenga este formato.

La parte resaltada en negrita será constante para nosotros, que solo empleamos la API *Place*, pues es la forma de invocar dicho servicio, lo que viene tras esto variará dependiendo de lo que deseemos obtener. Para nuestra aplicación necesitaremos realizar dos peticiones por local. Una para solicitar el "Place ID", esto es un identificador único del local dentro de Google Maps, y, una vez obtenido este ID del local, necesitaremos otra petición para solicitar los comentarios de dicho local. Mostramos a continuación las peticiones que necesitamos.

La primera petición será para solicitar el "Place ID", tal como vemos en la siguiente ilustración.

`https://maps.googleapis.com/maps/api/place/findplacefromtext/output?parameters`

Ilustración 15. Formato URL para solicitar el "Place ID" de Google Maps

Donde empleamos *"findplacefromtext*" este servicio nos permite realizar una búsqueda en Maps empleando el número de teléfono, una dirección o un nombre. En nuestro caso emplearemos el número de teléfono para buscar los locales. El número de teléfono deberá de estar en formato internacional, precedido por el signo "+" y seguido del código del país (+34 en el caso de números españoles). La codificación en el formato requerido para una URL sería "=%2B34" donde "%2B" representaría el signo "+". Por tanto, para la petición que necesitamos, en la parte de parámetros pondremos los siguiente:

`/json?input=%2B34`**`numTelefono`**`&inputtype=phonenumber&fields=place_id&key=`**`mykey`**

Ilustración 16. Parámetros necesarios para solicitar el "Place ID".

Donde "numTelefono" será el número del local del que deseamos obtener el "Place ID" y" mykey" es nuestra

clave API.

La segunda petición la empleamos para solicitar los comentarios, y será como mostramos en la ilustración inferior:

```
https://maps.googleapis.com/maps/api/place/details/output?parameters
```

Ilustración 17. Formato de la URL para solicitar los cometarios.

En esta segunda petición empleamos el "Place ID" obtenido en la primera para solicitar los *reviews*, esto lo haremos empleando los parámetros, los cuales sustituiremos por los siguientes:

```
https://maps.googleapis.com/maps/api/place/details/json?place_id=
placeId&fields=review&key=mykey
```

Ilustración 18. Parámetros necesarios para solicitar los comentarios.

Donde "placeId" será la que obtuvimos en la petición anterior y "mykey" vuelve a ser nuestra clave API.

Con esto hemos visto todo lo que necesitamos para poder emplear los servicios de *Place API* que requerimos en nuestra aplicación, ahora pasaremos a estudiar lo que necesitamos de Microsoft Azure.

4.3 Azure

En este caso lo único que requerimos instalar en nuestro ordenador serán las librerías que Azure tiene para Python. Con estas librerías vamos a poder comunicarnos con los servicios de Azure empleando peticiones *REQUEST*. Los requisitos completos los podemos encontrar en la documentación de la API [22] en dicha documentación están recogidos los pasos necesarios para usar Text Analytics en un ordenador con Windows. A continuación, resumiremos lo que hemos realizado nosotros.

Lo primero que necesitamos es crearnos una cuenta de Azure, en nuestro caso hemos empleado el correo de estudiante de la Universidad de Sevilla y hemos creado una cuenta de estudiantes en Azure. Esta cuenta nos permitirá emplear muchos de los servicios de Azure de forma gratuita durante doce meses, con límites de uso que varían según el servicio. En el caso de Text Analytics tiene un límite de 5000 mil peticiones por mes.

Tras crear la cuenta procedemos a instalar la librería de Text Analytics para Python. Esta librería contiene los métodos necesarios para emplear la API en Python, para instalarla empleamos la siguiente sentencia, que nos instalará la última versión estable, la 3.0 en nuestro caso.

```
pip install –upgrade azure-ai-textanalytics
```

Ilustración 19. Instalación librería Text Analytics.

Tras esto ya podremos invocar los servicios de Text Analytics. Aunque para poder usarlos correctamente requeriremos de unas claves. Estas claves las generamos en Azure Portal donde deberemos crear un nuevo recurso. Una vez hemos creado el nuevo recurso creamos la clave de API, esto lo hacemos clicando en "Claves y punto de conexión" tal y como mostramos en la siguiente imagen:

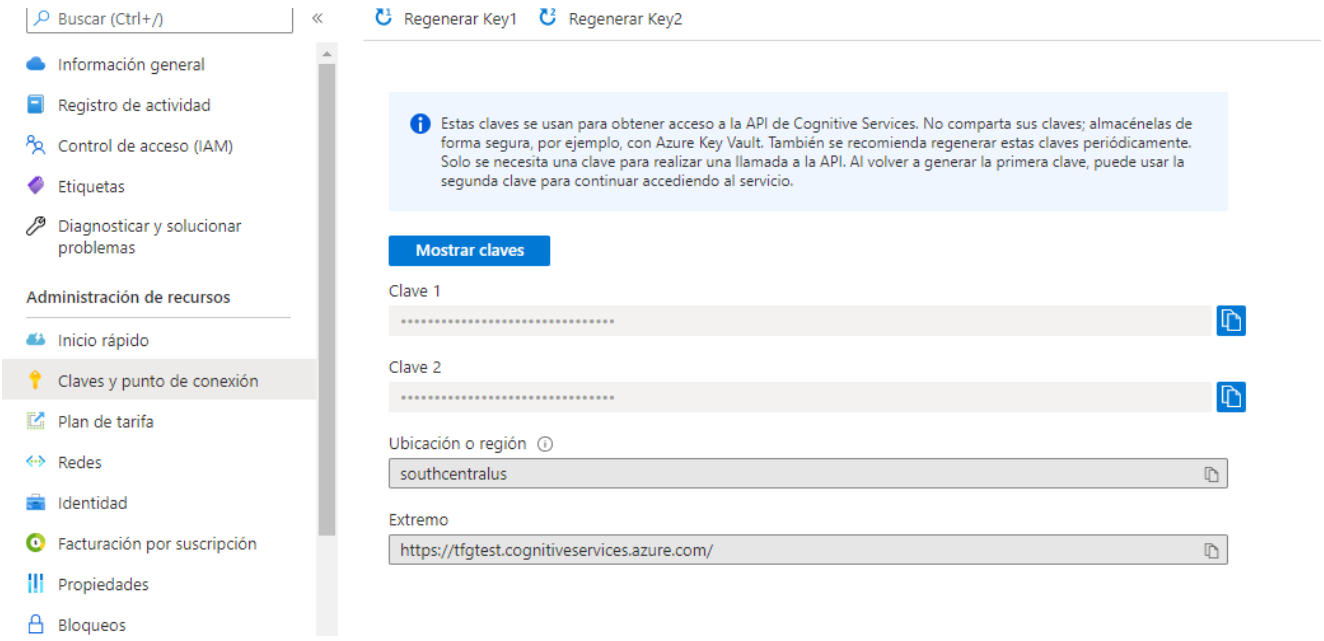


Ilustración 20. Menú "Claves y conexión".

En el menú que mostramos en la figura superior tenemos todo lo necesario para usar la API web, además en la documentación [18] de la API tenemos ejemplos de cómo realizar la autenticación.

Con esto ya estamos en disposición de emplear la API de Text Analytics, en este caso deberemos de autenticarnos cada vez que queramos realizar el análisis, para ello usamos el método que está recogido en la documentación oficial y que explicaremos más detalladamente más adelante.

Tras completar los procesos mostrados desde el comienzo del capítulo, estaremos en disposición de poder emplear los servicios presentados. Tras esto podemos mostrar nuestro escenario de pruebas, cuya implementación procedemos a explicar a continuación. La explicación la dividiremos en fases, igual que en el punto anterior, cada fase la realizará un método diferente.

4.4 Fase inicial

En esta fase obtendremos los números de teléfono de los locales que queremos analizar de un fichero de texto. Este fichero llamado “numTelefonos.txt” contiene un número de teléfono por línea. De esta forma se analizarán tantos locales como líneas tenga el fichero.

El código que se encarga de esto está en el fichero "main.py", en el cuál tenemos definido el método `main`, que mediante un bucle va leyendo y realizando llamadas al método `google` del fichero” google.py”, pasándole varios argumentos:

- ***fAWS***: descriptor de fichero donde guardaremos las puntuaciones, *scores*, que nos llegaran en las respuestas del análisis de sentimientos realizado con AWS Comprehend.
- ***fAzure***: igual que el anterior, pero para el análisis que realizamos con Azure Text Analytics.
- ***numTelefono***: aloja el número de teléfono actual.
- ***indice***: variable que empleamos como índice.

Como respuesta recibe una lista con los resultados del análisis, esta lista se guarda en un diccionario llamado `diccionarioFinal` cuya clave es el número de teléfono del local.

```
from google import google
from visionado_html import visionado_html

fAWS=open('OpinionAWS.txt','w')
fAzure=open('OpinionAzure.txt','w')
"""
*Mediante un bucle for leeremos los números y realizaremos el análisis
"""
indice=0
diccionarioFinal={}
for entity in numFile.readlines():
    analisisFinal = google(fAWS,fAzure,numTelefono=entity)
    diccionarioFinal[entity]=(analisisFinal)
    indice+=1
    #Aquí ahora mismo cada número de teléfono tiene los dos análisis
fAWS.close()

fAzure.close()
```

Ilustración 21. Llamada a `google`.

Cuando terminan las llamadas al método `google`. Realizamos una llamada al método `visionado_html`. Le pasamos como parámetro el objeto `diccionarioFinal` comentado antes, con el objetivo de representar los datos en un documento HTML. En caso de fallo mostramos un mensaje de error. En la siguiente ilustración podemos ver lo que acabamos de explicar.

```
resultado=visionado_html(diccionarioFinal)
if resultado!=1:
    print("Ha ocurrido un error al generar el HTML")
else:

    print("HTML generado con éxito. Fin del Análisis")
```

Ilustración 22. Llamada a `visionado_html`.

Pasamos ahora a explicar la fase de recolección de datos.

4.5 Fase de recolección de datos.

Esta fase se codifica en el archivo "google.py". En esta realizamos peticiones a Google Maps. Empleando una URL que montamos acorde a lo explicado en el apartado "Servicios de Google Maps" con el número de teléfonos que recibimos como parámetro. Necesitamos dos peticiones por local para poder acceder a los datos, en la primera realizamos una petición con el número de teléfono del local para poder obtener el "place_id" de este. En la segunda petición emplearemos el "place_id" para obtener los comentarios de los locales.

En ambas peticiones usamos la API *Place*, procederemos a explicar más a fondo como montamos las dos peticiones. Para ello sustituiremos los parámetros explicados antes por los que tengamos en cada momento.

La estructura de la URL que empleamos en la primera petición es la siguiente:

```
url1="https://maps.googleapis.com/maps/api/place/findplacefromtext/json?inpu
t=%2B34"+numTelefono+"&inputtype=phonenumber&fields=place_id&key="+mykey
```

Ilustración 23. URL para solicitar el "Place ID".

Donde "numTelefono" es el número de teléfono del local que queremos analizar, será un *string* que contenga el número, sin espacios ni caracteres especiales (ejemplo:666666666). Luego, "mykey" es nuestra API KEY. De esta forma usamos la misa estructura de URL para todas las peticiones, únicamente cambiamos el valor del número de teléfono. Se observa que estamos solicitando la "place_id" en la parte "fields=place_id".

La respuesta nos llega en formato *JSON*. Una vez finalizada la primera petición, procedemos a realizar la segunda, ya con el objetivo de obtener comentarios de usuarios, "reviews". La estructura de la URL es la siguiente:

```
url12="https://maps.googleapis.com/maps/api/place/details/json?place_id="+ap
iKey+"&fields=review&key="+mykey
```

Ilustración 24. URL para solicitar los comentarios de usuario.

Como se puede ver en este caso empleamos la "place_id" para realizar la petición y tenemos marcado que queremos los comentarios (reviews). Esto no es más que la parte "fields=review". Al igual que antes tenemos que añadir nuestra API KEY.

```
response12 = requests.get(url12)
rev_bruto=response12.json()
```

Ilustración 25. Obtener los comentarios

En la ilustración superior observamos como guardamos los comentarios, o reviews, recibidos en una lista llamada "rev_bruto". Esta lista contendrá cinco comentarios en total, que es el número de comentarios que devuelve una petición. Esto facilitará el análisis en la siguiente fase. Con esto hemos concluido la primera fase, en la que hemos recopilado la información que queremos analizar. Tras esto llamamos a `analisisAWS` y a `analisisAZURE`, donde realizamos los análisis. Les pasamos como argumentos el número de teléfono del local, una lista con sus comentarios y el descriptor de archivo que explicamos en la fase inicial.

```
#Llamamos a los métodos de análisis.
listaResultadoAWS=analisisAWS(numTelefono,rev_bruto,fAWS)
listaResultadoAzure=analisisAzure(numTelefono,rev_bruto,fAzure)

analisis={'AWS':listaResultadoAWS,'AZURE':listaResultadoAzure}


return analisis
```

Ilustración 26. Llamada a `analisisAWS` y `analisisAzure`

Con esto terminamos esta fase de recolección de datos y pasamos a la de análisis. Este método devolverá una lista con el resultado del análisis que agrupamos en un diccionario de nombre "análisis" que devolvemos, tal y como se puede observar en la ilustración superior.

4.6 Fase de análisis

Esta fase está codificada en el fichero "analisisAWS.py" y "análisisAzure.py". En ambas aplicamos el análisis empleando *machine learning*, empleando la opción de AWS y de Azure respectivamente. A continuación, describimos las consultas y el análisis que hemos realizado, comenzaremos con la opción de AWS tras completarla comentaremos las diferencias que tiene la opción de Azure.

4.6.1 Análisis empleando AWS Comprehend

Lo primero que hemos hecho en esta fase es establecer la conexión con AWS Comprehend. Esto se hace mediante la librería boto3.

```
client=boto3.client('comprehend')
```

Ilustración 27.Conectar con AWS Comprehend.

Con esta sentencia hemos establecido una conexión con AWS, para ello previamente se han realizado los pasos que se explicaron en el capítulo anterior.

En esta fase se realizan dos análisis. El primero de ellos es un análisis del lenguaje, realizado para determinar en qué idiomas están escritos los comentarios. Como nosotros estamos realizando una prueba de concepto nos hemos limitado a contemplar dos idiomas, español e inglés. Este análisis lo hemos realizado empleando el método `detect_dominant_lenguage` el cual solicita un único parámetro, "TextList" que no es más que una lista de *string* que debe de contener el texto que queremos analizar. La llamada se realiza tal y como mostramos en la siguiente imagen:

```
for entity in listaComentarios["result"]["reviews"]:
    usuPunt.append(entity["rating"])
    responseIdioma=client.detect_dominant_language
        (Text=str(entity["text"]))

    aux2=responseIdioma.get('Languages')
```

Ilustración 28. Llamada método detect_dominant_Lenguage

Este método nos devuelve un objeto *dict* con dos campos *ResultList* y *ErrorList*, procedemos a explicar la primera que es la que debemos emplear [10]:

- ***ResultList***: objeto tipo lista que contiene los resultados de la operación. Los resultados se clasifican en orden ascendente por el campo índice y coinciden con el orden de los documentos en la entrada. Si alguno de los documentos contiene un error, esta lista quedaría vacía. Esta a su vez tiene los siguientes campos:
    - ***Index***: objeto Integer, es el índice de base cero del documento en la lista de entrada.
    - ***Laguages***: objeto tipo lista. En ella se guardan los idiomas dominantes que se han detectado junto a su nivel de fiabilidad.

Una vez sabemos todo esto emplearemos un bucle *for in* para recorrer el diccionario e ir separando los comentarios en listas diferentes en función a su idioma, por lo que tendremos dos listas diferentes, "listaIngles" y "listaCastellano". Aprovechamos el bucle para guardar la puntuación que el usuario ha dejado en Google Maps, la almacenamos en una lista llamada "usupunto", la usaremos para comprobar la fiabilidad de nuestros análisis.

Tras completar el primer análisis, procedemos con el segundo. Antes de comenzar con este análisis comprobamos que las listas de los idiomas contemplados no estén vacías. Tras esta comprobación realizamos la llamada al método tal y como se muestra a continuación:

```
#Comprobamos que hay comentarios en alguno de los dos idiomas
    if numComentariosCastellano !=0 or numComentariosIngles !=0:
        #Primero analizamos los comentarios en ingles
        for entity in listaIngles :
            responseSentimentIngles = client.detect_sentiment(Text=entity,
                                                    LanguageCode='en')
```

Ilustración 29. Llamada método `detect_sentiment`.

En esta ocasión empleamos el método `detect_sentiment` [11] el cual necesita dos parámetros:

1. ***TextList***: es el mismo que en el caso anterior
2. ***LanguageCode***: es un *string* con el que se le dice el idioma en el que está escrito el texto contenido en *TextList*.

Destacamos que el segundo parámetro es un *string*, pero únicamente acepta una serie de idiomas, identificados por unas letras específicas. Por ejemplo, para el español sería "es" y en inglés "en" [12].

La respuesta que recibimos es similar a la anterior, solo que ahora en *ResultLis* nos encontramos los siguientes campos:

- ***Index***: Idéntico al del campo anterior.
- ***Sentiment***: de tipo string. Aquí se guarda el sentimiento detectado. Como ya se ha comentado puede ser *positive*, *negative*, *mixed* o *neutral*.
- ***SentimentScore***: objeto tipo dict que nos indicara el nivel de confianza de los resultados para cada opción de sentimientos, por lo que se compone de cuatro campos más, todos de tipo Float:
    - ***Positive***: nos indica el nivel de confianza que Comprehend tiene de la precisión de su detección para el sentimiento *positive.*
    - ***Negative***: nos indica el nivel de confianza que Comprehend tiene de la precisión de su detección para el sentimiento *negative.*
    - ***Mixed***: nos indica el nivel de confianza que Comprehend tiene de la precisión de su detección para el sentimiento *mixed.*
    - ***Neutral***: nos indica el nivel de confianza que Comprehend tiene de la precisión de su detección para el sentimiento *neutral.*

Por tanto, para completar este segundo análisis realizaremos a lo sumo, dos peticiones por local. Una para comentarios en español y otra para comentarios en inglés, si contemplásemos más idiomas pues se añadirían sus peticiones en ambas fases del análisis. Como ya se ha explicado, esto nos da como respuesta una catalogación en *positive*, *negative*, *mixed* o *neutral.*

Para poder guardar los datos se va recorriendo la respuesta con un bucle *for in*, igual que realizamos en el primer análisis. En dicho bucle, además, vamos escribiendo los datos del análisis en un fichero de texto que recibimos por parámetro en la llamada al método. Lo que escribimos es lo que podemos observar en la siguiente ilustración.

```
for entity in listaIngles :
    responseSentimentIngles = client.detect_sentiment(Text=entity,
                                            LanguageCode='en')
    f.write("Numero de teléfono = "+str(indice)+"\n")
    f.write("\nEl comentario analizado es:"+entity+"\n")
    f.write("\nDocument Sentiment by AWS in English: {}".
        format(responseSentimentIngles["Sentiment"]))
    f.write("\n")
    f.write("Overall scores:\n positive={0:.6f}\n neutral={1:.6f}\n
        negative={2:.6f}\n mixed={3:.6f} \n".format(
        responseSentimentIngles["SentimentScore"]["Positive"],
        responseSentimentIngles["SentimentScore"]["Neutral"],
        responseSentimentIngles["SentimentScore"]["Negative"],
        responseSentimentIngles["SentimentScore"]["Mixed"],
    ))
    f.write("\n")
```

Ilustración 30. Proceso de escritura en "OpinionAWS.txt".

Como vemos escribimos el número de teléfono del local al que se le está realizando el análisis y los resultados detallado de dicho análisis. Esto lo hacemos con el objetivo de poder comparar el funcionamiento de ambos proveedores, como mostraremos en un capítulo posterior.

Una vez terminamos el bucle tenemos el número de veces que coincide cada sentimiento almacenado en variables. También guardamos el número de comentarios en cada uno de los idiomas contemplados.

Todas las variables usadas para contabilizar se almacenan en listas. Las relacionadas con los sentimientos van en *listaSentimiento*. La de los idiomas en *listaIdioma*.

Por último, empezamos a organizar los datos de cara a la siguiente fase. Para esto, creamos una variable tipo diccionario, llamada *analisisCompleto*, que contendrá los siguientes campos:

- ***analisisIdioma***: de tipo lista en la que almacenamos la lista creada con los contadores de idiomas contemplados.
- ***analisisSentimiento***: de tipo lista en la que almacenamos la lista creada con los contadores de los sentimientos.
- ***raitingUsuMean***: de tipo entero, contiene la media de las puntuaciones que dejan los autores de los comentarios en Google Maps. El objetivo de esta variable es de ayudarnos a validar los resultados de los análisis que hacemos durante las pruebas.
- ***numTotalComentarios***: de tipo entero que contiene el número total de comentarios analizados.
- ***ErrorGraficas***: nos indicará si existe un error en las gráficas que se generarán posteriormente. Por defecto valdrá cero, todo correcto, se le dará el valor uno en caso de que exista un error.
- ***PuntMedia***: de tipo entero donde almacenaremos la puntuación media que ha obtenido el local.

Una vez hemos completado los análisis y almacenado los datos calculamos la puntuación media del local. Esto lo hacemos asignándole unos valores de referencia a cada sentimiento, los valores son los siguientes:

Tabla 6. Valores para la puntuación media de los locales.

Positive	5
Negative	-1
Mixed	3.5
Neutras	2.5

De esta forma calculamos una puntuación media que nos sirve de ejemplo, tal y como se muestra en la siguiente figura:

```
puntPositive=5*analisisCompleto['AnalisisSentimiento'][0][0]
puntNegative=1*analisisCompleto['AnalisisSentimiento'][0][1]
puntMixed=3.5*analisisCompleto['AnalisisSentimiento'][0][2]
puntNeutral=2.5*analisisCompleto['AnalisisSentimiento'][0][3]

#Ya tenemos asignada las puntuaciones.
#La puntuación media del local será
#La suma de todas divido entre el número total de comentarios
puntMedia=(puntPositive-puntNegative+puntNeutral+
           puntMixed)/analisisCompleto['numTotalComentarios']
#Si solo existen comentarios negativos capamos a cero.
if puntMedia<0:
    puntMedia=0

analisisCompleto['PuntMedia']=puntMedia
```

Ilustración 31. Cálculo de puntuación media.

Observamos que truncamos a cero en caso de que la puntuación media fuese negativa.

Tras esto, se llamará al método `grafica`. Le pasamos como argumentos el diccionario *analisisCompleto*, un estero que cuando vale uno indica que la llamada la hace AWS o Azure en caso contrario y la variable "índice" que contiene el número de teléfono del local actual para seguir con la siguiente fase.

Hay que destacar que `grafica` devolverá un cero si se completó con éxito, lo cual provoca que se imprima un mensaje de error por la salida estándar si no es así. En cualquier caso, `analisisAWS` devuelve el diccionario "analisisCompleto".

```
if grafica(indice,analisisCompleto,1)!=0:
    print("Ha ocurrido un error al generar la gráfica\n")

    analisisCompleto['ErrorGraficas']=1
```

Ilustración 32. Error al generar la gráfica.

En caso de que las listas con los idiomas contemplados estuvieran vacías rellenamos el diccionario final con un texto de error e imprimimos un mensaje de error tal y como mostramos en la imagen.

```
print("Error no hay comentarios para el teléfono: "+str(indice))
listaIdioma.append("Error")

listaSentimineto.append("Error")
```

Ilustración 33. Error no existen comentarios.

En cualquier caso, la función `analisisAWS` devolverá *analisisCompleto* con el resultado del análisis o con los errores que hayan podido ocurrir.

4.6.2 Análisis empleando Azure

En esta opción realizamos los mismos análisis que en la anterior, siguiendo la misma estructura que en el caso de AWS. La diferencia está en la forma de llamar a los métodos de la API y en la forma de autenticarnos para poder usarla. Explicaremos como hacemos esto en la opción que nos da Microsoft.

Para autenticarnos empleamos el código de ejemplo que nos da Microsoft en la documentación oficial, tal y como comentamos en un apartado anterior, este método esta codificado de siguiente forma:

```
# use this code if you're using SDK version is 5.0.0
from azure.ai.textanalytics import TextAnalyticsClient
from azure.core.credentials import AzureKeyCredential
key="AQUÍ_VA_LA_CLAVE_PRIVADA_QUE_NOS_IDENTIFICA"
endpoint="https://NOMBRE_DEL_PROYECTO.cognitiveservices.azure.com/"

def authenticate_client():
    ta_credential = AzureKeyCredential(key)
    text_analytics_client = TextAnalyticsClient(
            endpoint=endpoint,
            credential=ta_credential)
    return text_analytics_client
```

Ilustración 34. Autenticación en Azure I.

Donde “key” y “endpoint” son las claves que explicamos anteriormente. Esta función la llamamos al principio del fichero `analisisAZURE`. Como podemos observar para emplear los servicios de Azure requerimos importar dos librerías, la primera para poder autenticarnos y la segunda para emplear la API de Text Analytics.

```
client = authenticate_client()
```

Ilustración 35. Autenticación en Azure II.

De esta forma ya estaríamos autenticados y podemos hacer uso de la API.

Como en el caso de AWS realizaremos dos análisis, todas las variables explicadas para AWS se aplican en Azure, la diferencia la encontramos en la forma de invocar la API. A continuación, mostramos como lo hacemos con Azure.

```
responseIdioma=client.detect_language(documents = documents,
                                      country_hint = '')[0]
aux2=responseIdioma.primary_language.name

if aux2=='English' :
    listaIngles.append(entity["text"])
    numComentariosIngles+=1
```

```
        else:
            if aux2=='Spanish' :
                listaCastellano.append(str(entity["text"]))
                numComentariosCastellano+=1
            else:
                print('El comentario no está ni en ingles ni en castellano')
```

Ilustración 36. Análisis de idioma con Azure.

Como podemos observar en la ilustración superior, Azure nos devuelve el idioma detectado en inglés y lo obtenemos mediante el método `primary_language.name`. Teniendo esto en cuenta hemos modificado la estructura para clasificar los comentarios correctamente. Queremos aclarar que el método `detect_language` tiene el parámetro "contry_hint" el cual se usa para eliminar la ambigüedad que pueda existir al detectar un idioma, para usarlo tenemos que especificar un código de país de dos letras, por defecto la API usa "US". Para eliminar este comportamiento se pone vacío, tal y como mostramos en el código de la ilustración superior.

Para el análisis de sentimiento invocamos al método `analyze_sentiment`, como mostramos en la siguiente ilustración.

```
        for entity in listaIngles :
            documents=[]
            documents.append(entity)
            responseSentimentIngles =client.analyze_sentiment(
                                                documents = documents)[0]
```

Ilustración 37. Análisis de sentimientos con Azure.

A diferencia de AWS, en Azure no es necesario especificar el idioma en el que esa escrito el texto que queremos analizar, por lo que solo le pasamos el comentario como argumento al método y este lo analiza. Al igual que en Comprehend, nos devuelve si el comentario es *positive*, *negative*, *neutral* o *mixed* junto al nivel de confianza de los resultados.

Otra diferencia la encontramos en que Azure posee métodos para acceder a la información de la respuesta, por lo que el acceso a la respuesta lo haremos empleando dichos métodos. En la siguiente imagen podemos apreciar los métodos que empleamos para escribir en el archivo "OpiniónAzure.txt" y apreciar las diferencias con el caso de AWS.

```
        for entity in listaIngles :
            documents=[]
            documents.append(entity)
            responseSentimentIngles =client.analyze_sentiment(
                                                documents = documents)[0]
            f.write("Numero de teléfono = "+str(indice)+"\n")
            f.write("\nEl comentario analizado es:"+entity)
            f.write("\nDocument Sentiment by AZURE in English: {}"
                    .format(responseSentimentIngles.sentiment)+"\n")
            f.write("Sentence
score:\nPositive={0:.6f}\nNeutral={1:.6f}\nNegative={2:.6f}\n".format(
               responseSentimentIngles.confidence_scores.positive,
               responseSentimentIngles.confidence_scores.neutral,
               responseSentimentIngles.confidence_scores.negative,

            ))
            f.write("\n")
```

Ilustración 38. Proceso de escritura en OpinionAzure.txt.

Con esto hemos terminado esta fase y pasamos a la fase de procesamiento de datos.

4.7 Fase de tratamiento de los datos

Esta fase está codificada en el fichero "grafica.py" donde hemos creado el método `gráfica`. Es en este método donde le daremos formato a los datos obtenidos en la fase anterior para poder interpretarlos de forma sencilla.

Para que la visualización sea ágil y simple generaremos unos diagramas de barras. Para esto emplearemos la librería *matplotlib.*

En esta fase usamos los datos obtenidos del análisis y que recibimos como parámetro para crear dos diagramas de barras por cada uno de los proveedores. Uno con el número de hablantes por idioma y otro representando el número de sentimientos de cada tipo.

Lo primero que hacemos es comprobar si la invocación al método `gráfica` proviene de `analisisAWS` o de `analisisAzure`, lo comprobamos mediante la siguiente sentencia:

```
def grafica(indice,analisisCompleto,aws):
    #Lo primero será preparar los datos que recibimos para graficarlos
    if(aws==1):
```

Ilustración 39. Comprobar si la llamada es desde AWS o desde Azure.

Tras comprobar para quién debemos crear las gráficas, las creamos y estás son guardadas en el directorio donde se ejecuta la aplicación. Para poder distinguir las gráficas de los diferentes locales, empleamos el número de teléfono del local y la fuente del análisis a la hora de nombrarlas. En la siguiente imagen vemos como lo hacemos.

```
plt.bar(idiomas, numHablantes)
indice=indice.strip('\n')
plt.savefig('grafica_Local_AWS_'+str(indice)+'_numHablantes.png')
```

Ilustración 40. Guardando gráfica análisis de idioma de AWS.

```
plt.bar(sentimientos, usuSentiment)
plt.savefig('grafica_Local_AWS_'+str(indice)+'_Sentimientos.png')
```

Ilustración 41. Guardando gráfica análisis de sentimientos de AWS.

Como vemos la variable índice contiene el número de teléfono, hemos puesto un ejemplo de AWS y otro de Azure para que se aprecie la nomenclatura que empleamos en ambos casos.

Una vez hemos guardado las gráficas devolvemos un cero indicando que todo ha ido bien y terminamos la fase. Pasamos a explicar la última fase, visionado de datos.

4.8 Fase de visionado de datos.

En esta fase generamos un fichero HTML a partir de una plantilla. El código está codificado en el fichero "visionado_html.py". A diferencia de los ficheros anteriores, está función se ejecuta una única vez, lo hace una vez se han analizado todos los locales, es llamada por el método explicado en la fase inicial.

En la plantilla encontramos los elementos mínimos para generar un fichero HTML, así como parámetros que sustituiremos para añadir la información. La plantilla está preparada para representar la información de tres locales. Para que funcione bien, `visionado_html` recibe el diccionario final del análisis que ya explicamos en la fase inicial. De este diccionario extraemos los números de teléfono, estos nos servirán para poder identificar los locales y para poder acceder a las gráficas creadas en la fase anterior.

A continuación, mostramos una serie de capturas donde se observa como rellenamos la plantilla con los datos de los locales analizados.

```
numeroTelefono1=dicaux[0]
numeroTelefono2=dicaux[1]
numeroTelefono3=dicaux[2]

puntMedia1=str(diccionarioVisionado[numeroTelefono1]['AWS']['PuntMedia'])
puntMedia2=str(diccionarioVisionado[numeroTelefono2]['AWS']['PuntMedia'])
puntMedia3=str(diccionarioVisionado[numeroTelefono3]['AWS']['PuntMedia'])

puntUsu1=str(diccionarioVisionado[numeroTelefono1]['AWS']['raitingUsuMean'])
puntUsu2=str(diccionarioVisionado[numeroTelefono2]['AWS']['raitingUsuMean'])
puntUsu3=str(diccionarioVisionado[numeroTelefono3]['AWS']['raitingUsuMean'])

puntMedia1Azure=str(diccionarioVisionado[numeroTelefono1]['AZURE']['PuntMedia'])
puntMedia2Azure=str(diccionarioVisionado[numeroTelefono2]['AZURE']['PuntMedia'])

puntMedia3Azure=str(diccionarioVisionado[numeroTelefono3]['AZURE']['PuntMedia'])
```

Ilustración 42. Diccionario para plantilla HTML I.

Como se observa en la ilustración superior creamos variables para almacenar los números de teléfono y las puntuaciones medias, tanto las obtenidas a partir de los análisis como las que provienen de Google Maps.

```
#CREAMOS UN DICCIONARIO
rutaIdioma1="../grafica_Local_AWS_"+numeroTelefono1+"_numHablantes.png"
rutaSentimiento1="../grafica_Local_AWS_"+numeroTelefono1+"_Sentimientos.png"
rutaIdiomaAzure="../grafica_Local_Azure_"+numeroTelefono1+"_numHablantes.png"
rutaSentimientoAzure="../grafica_Local_Azure_"+numeroTelefono1+"_Sentimientos.png"

rutaIdioma2="../grafica_Local_AWS_"+numeroTelefono2+"_numHablantes.png"
rutaSentimiento2="../grafica_Local_AWS_"+numeroTelefono2+"_Sentimientos.png"

rutaIdiomaAzure2="../grafica_Local_Azure_"+numeroTelefono2+"_numHablantes.png"
rutaSentimientoAzure2="../grafica_Local_Azure_"+numeroTelefono1+"_Sentimientos.png"

rutaIdioma3="../grafica_Local_AWS_"+numeroTelefono3+"_numHablantes.png"
rutaSentimiento3="../grafica_Local_AWS_"+numeroTelefono3+"_Sentimientos.png"

rutaIdiomaAzure3="../grafica_Local_Azure_"+numeroTelefono3+"_numHablantes.png"

rutaSentimientoAzure3="../grafica_Local_Azure_"+numeroTelefono3+"_Sentimientos.png"
```

Ilustración 43. Diccionario para plantilla HTML II.

En la ilustración supero podemos ver como creamos las variables que alojan las rutas de las gráficas. Empleamos los números de teléfono de los locales para poder crear dicha ruta. Tras esto creamos un diccionario “d”, que mostramos en la siguiente ilustración.

```
d={ 'prueba1':numeroTelefono1,'puntUsu1':puntUsu1,'puntMedia1':puntMedia1,
   'rutaIdioma1prueba':rutaIdioma1,
   'rutaSentimiento1':rutaSentimiento1,'puntMediaAzure1':puntMedia1Azure,
   'rutaIdioma1pruebaAzure':rutaIdiomaAzure,
   'rutaSentimientos1pruebaAzure':rutaSentimientoAzure,
   'prueba2':numeroTelefono2,
   'puntUsu2':puntUsu2,
   'puntMedia2':puntMedia2,'rutaIdioma2':rutaIdioma2,
   'rutaSentimiento2':rutaSentimiento2,'puntMediaAzure2':puntMedia2Azure,
   'rutaIdioma2pruebaAzure':rutaIdiomaAzure2,
   'rutaSentimientos2pruebaAzure':rutaSentimientoAzure2,
   'prueba3':numeroTelefono3,'puntUsu3':puntUsu3,
   'puntMedia3':puntMedia3,'rutaIdioma3':rutaIdioma3,
   'rutaSentimiento3':rutaSentimiento3,'puntMediaAzure3':puntMedia3Azure,
   'rutaIdioma3pruebaAzure':rutaIdiomaAzure3,

   'rutaSentimientos3pruebaAzure':rutaSentimientoAzure3}

#Sustituimos la variable de Python por la que queremos que aparezca en HTML

result=src.substitute(d)
```

Ilustración 44. Diccionario para plantilla HTML III.

Las variables entrecomilladas son los parámetros de la plantilla HTML que hemos creado mientras que las no entrecomilladas son las variables que contienen los datos que queremos incluir en dicha plantilla. Empleando el método `substitute` sustituiremos los parámetros de la plantilla por los que deseamos.

Con esto hemos completado nuestra prueba. Quedando como resultado un documento HTML llamado "análisis", que veremos detalladamente en el próximo capítulo. En el que encontramos la información que hemos generado de forma clara y detallada.

4.9 Conclusión

Como se ha podido observar, hemos podido realizar varios análisis empleando *machine learning* usando servicios de proveedores en la nube, mediante sus APIs. De forma automática hemos podido realizar dos clasificaciones de los comentarios extraídos de Google Maps.

- La primera en función al idioma, obteniendo de esta forma información sobre la procedencia de nuestros clientes.
- La segunda, del sentimiento que expresa su comentario lo que nos proporciona información sobre cómo se han sentido en nuestro establecimiento.

Todo esto lo hemos logrado empleando servicios en la nube y un ordenador doméstico, demostrando que está tecnología no requiere grandes inversiones en hardware ni conocimientos profundos en *machine learning*. En el siguiente capítulo mostraremos y comentaremos los resultados de las pruebas que hemos realizado.

5 PRUEBAS Y FUNCIONAMIENTO

Este capítulo lo dedicamos a mostrar las pruebas que hemos realizados, así como a mostrar los resultados de dichas pruebas.

Para las pruebas hemos empleado restaurantes, los cuales se han seleccionado de forma aleatoria. El número de comentarios analizado por local es el que devuelve Google Maps en una única petición.

En nuestra prueba realizamos peticiones para una serie de locales identificados por el número de teléfono que tienen publicado en Google Maps. Estos teléfonos se almacenan en un fichero de texto, de esta forma modificando dicho fichero podremos añadir o eliminar locales.

Resumiendo, para las pruebas hemos realizado una única petición a Maps por local, lo que nos deja cinco comentarios por local. Al realizar los análisis explicados en los capítulos 3 y 4 obtenemos gráficas como las siguientes, que son el resultado de los análisis realizados sobre un único local:

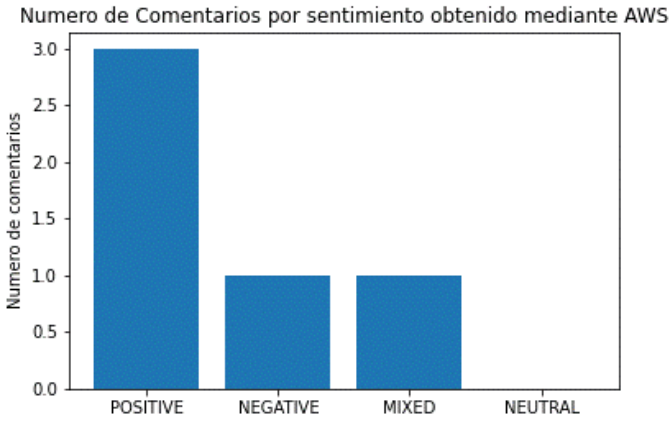


Ilustración 45. Número de comentarios por sentimiento

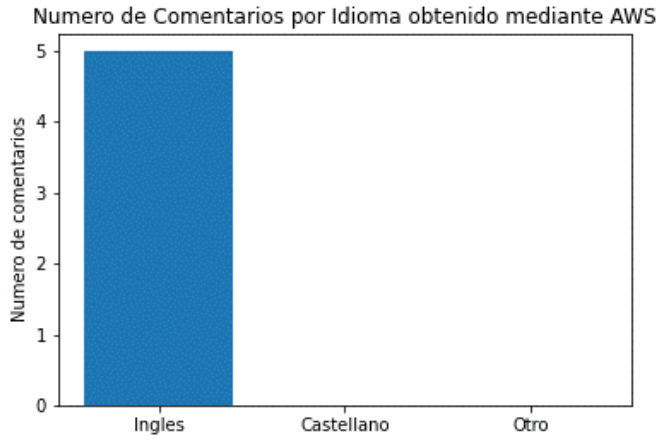


Ilustración 46. Número de hablante por idioma

Las gráficas superiores corresponden a los dos análisis que realizamos sobre los comentarios. Procedemos a comentar cada una de ella con más detalle:

- La primera gráfica corresponde al análisis de los sentimientos, se obtiene tras aplicar el método `client.detect_sentiment` de Comprehend a cada uno de los comentarios. De esta gráfica

podemos extraer información sobre cuál ha sido la experiencia general de los clientes. Ya que una buena experiencia se traduce en un comentario positivo, esto quiere decir que el sentimiento encerrado tras el comentario es bueno, una experiencia ni demasiado buena ni demasiado mala correspondería con un sentimiento neutral o mixed y una mala experiencia nos arrojaría un comentario con un sentimiento negativo. Como se observa en la gráfica el sentimiento predominante en el local analizado es el positivo, que acapara tres de los cinco comentarios analizados. Mientras que negativo y mixed solo tenemos uno de cada. Esto nos viene a decir que el restaurante está bien valorado, la mayoría de sus clientes salen contento, alguno sale neutro y solo uno sale con un mal sentimiento.

- La segunda de ella corresponde con el análisis del idioma. Se obtiene tras aplicar el método `client.detect_dominant_language` de AWS Comprehend sobre cada uno de los comentarios extraídos. Como se puede observar vemos que, en este caso, se obtiene que todos los comentarios son en inglés por lo que podemos deducir que una mayoría de los clientes de este restaurante son de origen inglesa o tienen el inglés como lengua principal. Por lo tanto, en este restaurante debería de haber trabajadores capacitados para atender en inglés. Esto no nos debe extrañar ya que viendo los comentarios directamente en la aplicación de Google Maps podemos observar como muchos están escritos en inglés.

No obstante, lo que obtenemos tras finalizar el escenario no son las gráficas sueltas. Obtenemos un documento HTML. En este documento podemos ver las gráficas de todos los locales identificados mediante su número de teléfono y la procedencia de la gráfica, si procede de AWS o de Azure. A continuación, mostramos dicho documento HTM de dos de las pruebas que hemos realizado para comentarlo con más detalle:

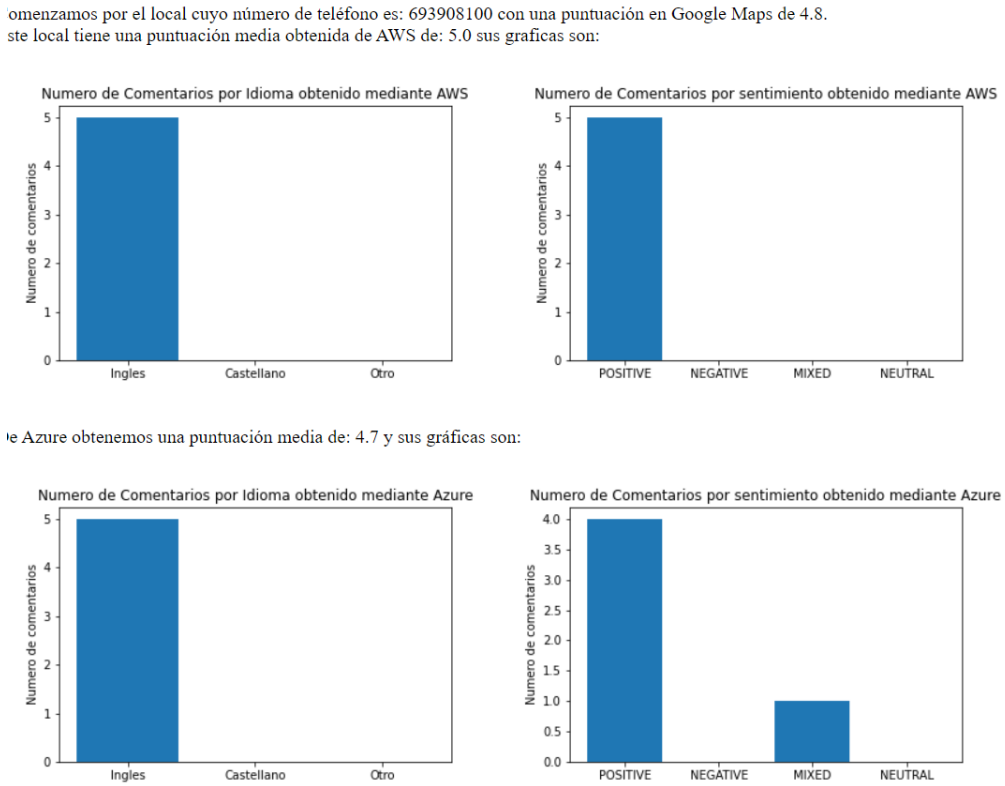
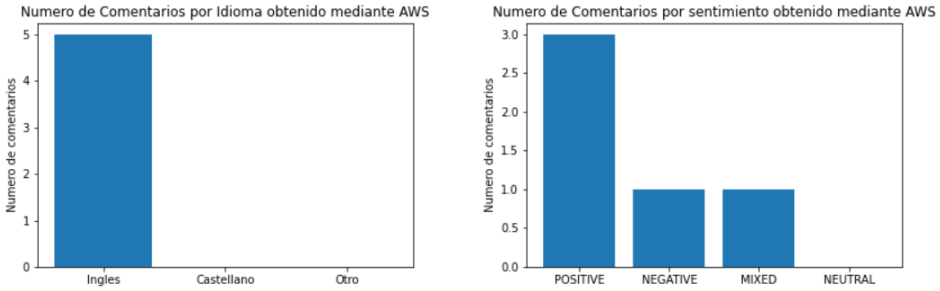


Ilustración 47. Fragmento I de la visualización final del análisis de la prueba I.

Continuamos con el local cuyo número de teléfono es: 956439183 con una puntuación en Google Maps de 3.6.
Este local tiene una puntuación media obtenida de AWS de: 3.5 y sus graficas son:



De Azure obtenemos una puntuación media de: 3.2 y sus gráficas son:

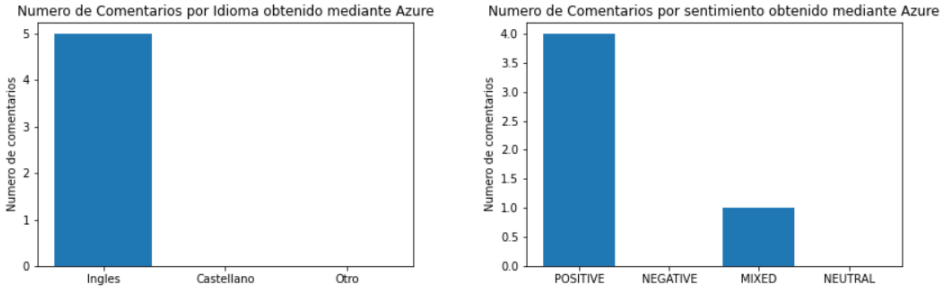
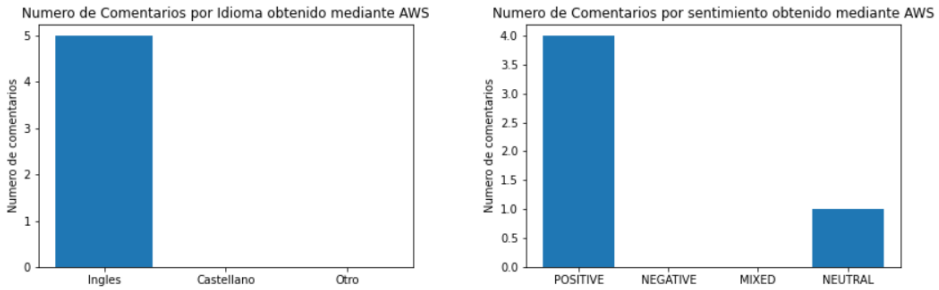


Ilustración 48. Fragmento II de la visualización final del análisis de la prueba I.

Por último el local cuyo número de teléfono es: 956439278, con una puntuación en Google Maps de 5.0.
Este local tiene una puntuación media de: 4.5 y sus graficas son:



De Azure obtenemos una puntuación media de: 5.0 y sus gráficas son:

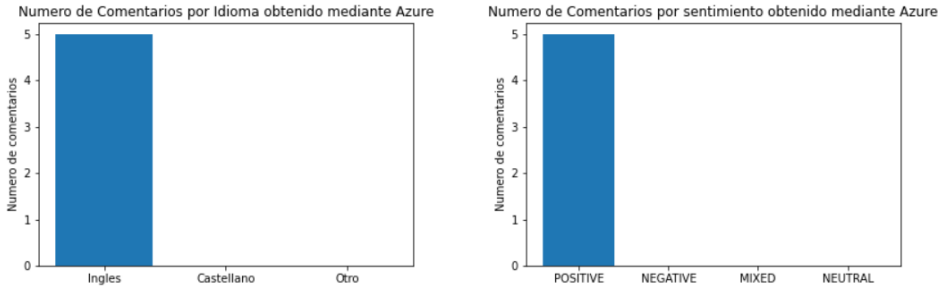
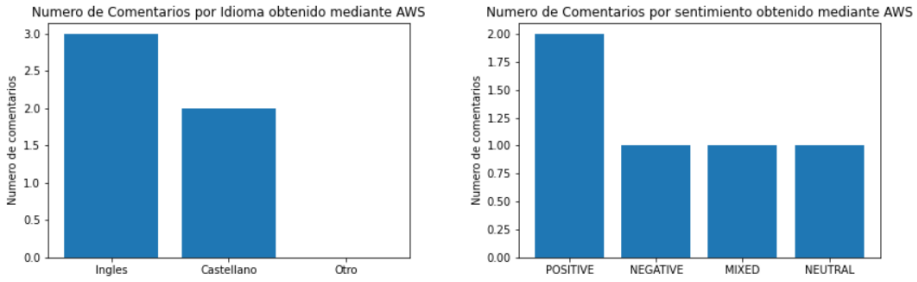


Ilustración 49. Fragmento III de la visualización final del análisis de la prueba I.

En las imágenes superiores observamos como presentamos la información. Como podemos ver, primero presentamos al local mediante su número de teléfono, añadiendo la puntuación media que le han dado los propios usuarios en Google Maps, obtenida como hemos comentado en puntos anteriores. Tras esto, la información obtenida de los análisis de AWS y luego de Azure.

Ahora mostramos los fragmentos de la otra prueba que hemos realizado con otros locales diferentes:

Comenzamos por el local cuyo número de teléfono es: 690954331 con una puntuación en Google Maps de 3.2.
Este local tiene una puntuación media obtenida de AWS de: 3.0 sus graficas son:



De Azure obtenemos una puntuación media de: 2.7 y sus gráficas son:

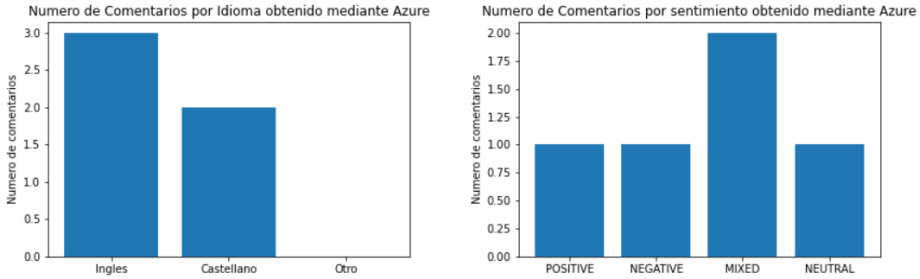
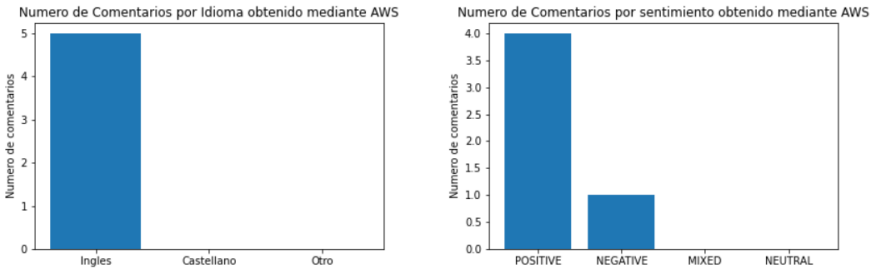


Ilustración 50. Fragmento I de la visualización final del análisis de la prueba II.

Continuamos con el local cuyo número de teléfono es: 956439332 con una puntuación en Google Maps de 4.6.
Este local tiene una puntuación media obtenida de AWS de: 3.8 y sus graficas son:



De Azure obtenemos una puntuación media de: 3.8 y sus gráficas son:

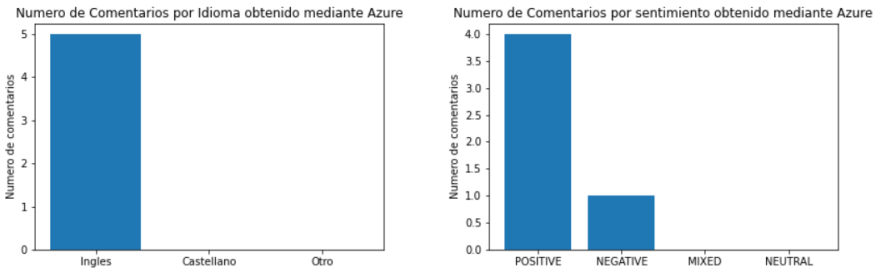


Ilustración 51.Fragmento II de la visualización final del análisis de la prueba II.

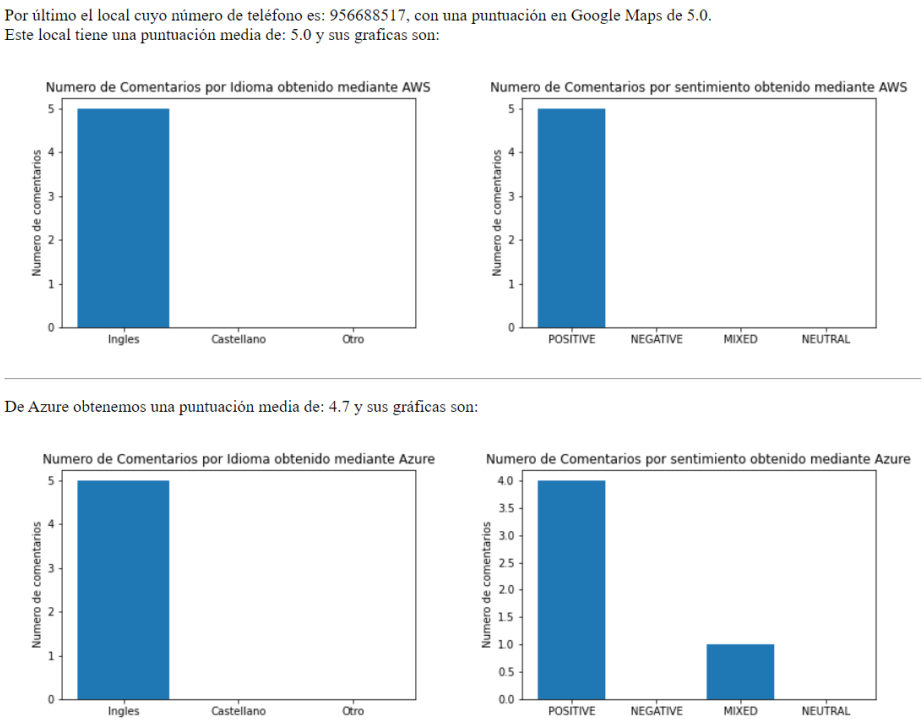


Ilustración 52. Fragmento III de la visualización final del análisis de la prueba II.

A continuación, vamos a explicar las diferencias que se aprecian entre los análisis de AWS y de Azure. Trataremos de analizar nosotros el comentario para comprar qué servicio se acerca mejor a lo que haría una persona.

5.1 Diferencias entre los análisis de AWS y los de Azure

Lo primero que vemos es que el análisis del idioma arroja los mismos resultados tanto en el caso de AWS como en el de Azure. Esto ha ocurrido en todas las pruebas que mostramos por lo que podemos concluir que ambos servicios cumplen con similar eficiencia está tarea.

Respecto al análisis de sentimientos no podemos decir los mismo, ya que con frecuencia los análisis con uno y otro proveedor nos arroja resultados diferentes, como podemos observar en las imágenes comentadas previamente. Podemos observar claramente como los resultados varían entre uno y otro análisis, para ver más claro porque ocurre esto consultamos los ficheros "OpinionAWS.txt" y "OpiniónAzure.txt" donde almacenamos los resultados de los análisis. A continuación, vamos a mostrar los *scores* de AWS y de Azure de los comentarios que arrojan análisis diferentes.

Comenzaremos comentado los resultados de la primera prueba donde los análisis de sentimientos arrojan un resultado diferente:

```
El comentario analizado es: Good food at a fair price. Apart from the meat and local fish they have
a wood burning pizza oven.

Worth a visit

Document Sentiment by AWS in English: POSITIVE

Overall scores:

positive=0.998149

neutral=0.001539

negative=0.000130

mixed=0.000182
```

Ilustración 53. Scores AWS.

```
El comentario analizado es: Good food at a fair price. Apart from the meat and local fish they have
a wood burning pizza oven.

Worth a visit

Document Sentiment by AZURE in English: mixed

Sentence score:

Positive=0.620000

Neutral=0.040000

Negative=0.340000
```

Ilustración 54. Scores Azure.

En las imágenes superiores podemos observar detalladamente como proporcionan la precisión del análisis que ambos servicios realizan. Con solo un rápido vistazo se comprueba que para un mismo comentario el resultado que arrojan ambos análisis es diferente. Vemos como AWS interpreta el comentario mostrado como positivo, con una certeza prácticamente absoluta, un 0.998149 sobre uno de puntuación. Por otro lado, Azure nos cataloga ese mismo comentario como *mixed* ya que su análisis arroja una puntuación de 0.62 para *positive* y de un 0.34 para *negative*.

Teniendo resultados dispares está claro que alguno de los dos análisis tiene que ser más certero que el otro. Leyendo el comentario vemos que, a priori, no hay palabras que contengan una clara connotación negativa. El comentario nos dice que hay buena comida y a un precio 'justo', entrecomillamos 'justo' ya que creemos que esa es la fuente de la diferencia. Azure interpreta esa palabra como algo con un sentimiento malo, negativo. Sin embargo, en el contexto de la oración, es fácil identificar que se refiere a que la comida tiene un precio adecuado, bueno para lo que se está ofreciendo. Por este motivo consideramos que el análisis de AWS es más cercano al que podemos hacer nosotros en este caso.

Este no es el único análisis en el que han diferido AWS y Azure, aquí podemos ver otro ejemplo.

```
El comentario analizado es: Best tuna we had in Zahara. Amazing dishes but we had to
order from the tapas menu. It gets crowded quite fast so better coming early to avoid
disapointment.

Document Sentiment by AWS in English: POSITIVE

Overall scores:

positive=0.990849

neutral=0.001402

negative=0.001181

mixed=0.006569
```

Ilustración 55. Scores AWS II.

```
El comentario analizado es: Best tuna we had in Zahara. Amazing dishes but we had to
order from the tapas menu. It gets crowded quite fast so better coming early to avoid
disapointment.

Document Sentiment by AZURE in English: mixed

Sentence score:

Positive=0.690000

Neutral=0.000000

Negative=0.310000
```

Ilustración 56. Scores Azure II.

En este caso, podemos comprobar que estamos ante una situación similar. AWS nos dice que el comentario es *positive* con una puntuación de 0.990849 mientras que Azure nos vuelve a comentar que es *mixed* con una puntuación para *positive* de 0.69 y de 0.31 para *negative*. Analizando nosotros la frase podemos comprobar que la palabra que, fuera del contexto de la oración, aporta un sentimiento negativo es *disappointment*. Es correcto

afirmar que esta palabra sola es negativa, pero en el contexto de la oración vemos que se refiere a que se debe llegar con tiempo a este restaurante para evitar largas esperas porque suele estar completo. No obstante, el autor del comentario deja ver que su experiencia ha sido positiva en general por lo que vemos exagerada la puntuación que Azure le da a *negative*. Por lo que volvemos a considerar más acertado el análisis realizado por AWS.

Como última diferencia en los comentarios que hemos analizamos lo siguiente.

```
El comentario analizado es: Top
Document Sentiment by AWS in English: NEUTRAL
Overall scores:
positive=0.398206
neutral=0.566581
negative=0.016047
mixed=0.019166
```

Ilustración 57. Scores AWS III.

```
El comentario analizado es: Top
Document Sentiment by AZURE in English: positive
Sentence score:
Positive=0.580000
Neutral=0.390000
Negative=0.030000
```

Ilustración 58. Scores Azure III.

Podemos ver como este comentario de una única palabra ha creado una situación donde, a pesar de que las puntuaciones de cada sentimiento no varían demasiado de un proveedor a otro, los resultados finales son bastante diferentes. Analicemos por partes.

En el caso de AWS tenemos las siguientes puntuaciones:

- *Positive*: 0.398206.
- *Neutral*: 0.566581.
- *Negative*: 0.016047

En el caso de Azure tenemos:

- *Positive*: 0.58.
- *Neutral*: 0.39.
- *Negative*: 0.03

Podemos ver claramente como AWS ha interpretado 'Top' como un sentimiento mayoritariamente *neutral*, en cambio Azure lo ha interpretado como *positive*. Lo curioso es que son prácticamente complementarios, con un 0.58 para *positive* de Azure y un 0.566581 *neutral* de AWS. En este caso consideramos que el análisis de Azure es más acertado que el de AWS.

A continuación, pasamos a analizar las diferencias en el análisis de sentimientos de la segunda prueba.

El primer comentario en el que difieren es el siguiente:

```
El comentario analizado es:
Un lugar muy agradable, como su personal.
La comida excelente, tanto presentación como sabor. Unas instalaciones limpias y sencillas. El servicio un poco lento, debido a que era la hora punta. Recomendable.
Document Sentiment by AWS in Spanish: NEUTRAL
Overall scores:
positive=0.042616
neutral=0.698480
negative=0.168553
mixed=0.090351
```

Ilustración 59. Scores AWS IV.

```
El comentario analizado es: Un lugar muy agradable, como su personal.
La comida excelente, tanto presentación como sabor. Unas instalaciones limpias y sencillas. El servicio un poco lento, debido a que era la hora punta. Recomendable.
Document Sentiment by Azure in Spanish: mixed
Sentence score:
Positive=0.710000
Neutral=0.060000
Negative=0.230000
```

Ilustración 60. Scores Azure IV.

Como podemos observar, AWS arroja un resultado *neutral* con un 0.042616 para *positive* y un 0.168553 para *negative*. Analizando nosotros el comentario podemos comprobar fácilmente que el análisis es erróneo, ya que la experiencia que nos relata el cliente es positiva. Elogia varios puntos del restaurante, el único punto negativo que expresa es el tiempo que se tardó en servir, aunque al mismo tiempo reconoce que se trataba de una hora punta. Por tanto, el análisis debería de tener una mayor puntuación *positive* que *negative*. En cambio, vemos como el análisis de Azure nos arroja un sentimiento *mixed*. En este análisis *positive* tiene una puntuación de 0.71 mientras que *negative* la tiene de 0.23. Un resultado acorde al análisis que hemos realizado nosotros. En este caso vemos como el análisis realizado por Azure es más cercano al nuestro que el de AWS.

```
El comentario analizado es:
Comida excelente, trato pésimo por parte de los jóvenes y poco profesionales camareros que se encararon con nosotros tras reclamarle el último plato y decirnos " que ya no se hacía más comida". Al decirles que no nos habían informado de que el último plato no iba a salir, el camarero dijo: " te voy a mandar a..." . El que parecía el jefe vino a restar importancia a la situación, mientras los tres camareros merodeaban alrededor de nuestra mesa soltando comentarios desafortunados. Finalmente, pusimos un poco de cordura a la desagradable situación hablando con el jefe o encargado y dejando de propina más de la merecida y por su parte cobrando todos y cada uno de los platos. Ni un chupito de invitación. Que normalmente se ofrece en la mayoría de los sitios. La educación se demuestra andando y la profesionalidad también.  Nunca mas.


Document Sentiment by AWS in Spanish: NEUTRAL
Overall scores:
 positive=0.042616
 neutral=0.698480
 negative=0.168553
 mixed=0.090351
```

Ilustración 61. Scores AWS V.

```
El comentario analizado es:

Comida excelente, trato pésimo por parte de los jóvenes y poco profesionales camareros
que se encararon con nosotros tras reclamarle el último plato y decirnos " que ya no se
hacía más comida". Al decirles que no nos habían informado de que el último plato no
iba a salir, el camarero dijo: " te voy a mandar a..." . El que parecía el jefe vino a
restar importancia a la situación, mientras los tres camareros merodeaban alrededor de
nuestra mesa soltando comentarios desafortunados. Finalmente, pusimos un poco de
cordura a la desagradable situación hablando con el jefe o encargado y dejando de
propina más de la merecida y por su parte cobrando todos y cada uno de los platos. Ni
un chupito de invitación. Que normalmente se ofrece en la mayoría de los sitios. La
educación se demuestra andando y la profesionalidad también.  Nunca mas.

Document Sentiment by Azure in Spanish: mixed

Sentence score:

Positive=0.370000

Neutral=0.120000

Negative=0.510000
```

Ilustración 62. Scores Azure V.

Como podemos observar en este caso AWS nos arroja un resultado *neutral*. Donde *positive* tiene una puntuación de 0.042616 y *negative* de 0.168553, por lo que nos muestra una experiencia más negativa que positiva. Analizando nosotros el comentario vemos que relata una mala experiencia, es más, el comentario finaliza con un "Nunca más". Esto nos bastaría para saber que el cliente no ha terminado contento con la experiencia ofrecida por el restaurante. Por lo que podemos afirmar que este análisis no es del todo correcto, intuimos que el error puede estar al inicio del comentario, donde dice "Comida excelente". Esto claramente tiene una connotación positiva, pero al leer el resto del comentario se ve claramente que la experiencia ha sido más negativa que positiva y no es esto lo que refleja el análisis de AWS. Por otro lado, el análisis de Azure nos arroja un resultado *mixed* en el que la puntuación de 0.37 para *positive* y de 0.51 para *negative* sí que refleja mejor la experiencia que nos relata el comentario. Por tanto, en este caso el análisis de Azure nos parece más acertado que el de AWS, sin llegar a considerarlo correcto, pues para nosotros la catalogación correcta sería *negative*.

```
El comentario analizado es:

Amazing food experience. I dont eat anything.

If it is something bad I'd not put it into my mouth.

This was exceptional. Especially the food, the service and the space were fine 4*

but the food is amazing

Document Sentiment by AWS in English: POSITIVE

Overall scores:

positive=0.984137

neutral=0.000902

negative=0.000415

mixed=0.014546
```

Ilustración 63. Scores AWS VI.

```
El comentario analizado es:
Amazing food experience. I dont eat anything.
If it is something bad I'd not put it into my mouth.
This was exceptional.
Especially the food, the service and the space were fine 4* but the food is amazing
Document Sentiment by AZURE in English: mixed
Sentence score:
Positive=0.730000
Neutral=0.020000
Negative=0.250000
```

Ilustración 64. Scores Azure VI.

En esta ocasión, observamos como AWS le da una calificación *positive* al restaurante, con una puntuación de 0.984137. Por otro lado, Azure le da una puntuación *mixed* con una puntuación de 0.73 para *positive* y de 0.25 para *negative*. Realizando nosotros el análisis de la frase podemos comprobar que, efectivamente, el sentimiento más probable que encierra el comentario es *positive*. Pensamos que la fuente del error puede estar en la palabra "bad" que por sí sola es mala, pero que en el contexto de la oración resulta ser positiva. Vemos como Azure ha vuelto a fallar a causa de no interpretar bien el contexto de la oración.

Con esto hemos terminado de comparar las diferencias encontradas entre los análisis de AWS y de Azure. A continuación, vamos a mostrar el precio que ha tenido los análisis realizados con AWS tras finalizar la prueba gratuita de 12 meses, en el caso de Azure no tenemos dichos datos pues la prueba no ha finalizado y en el caso de Google Maps tenemos un crédito mensual que tampoco hemos superado en nuestras pruebas.

5.2 Coste de las pruebas

Para poder verlo con más claridad mostramos un resumen de unas de las facturas, extraído del panel de facturación de la consola de AWS.

Ilustración 65. Resumen factura agosto 2021.

Como podemos ver se nos ha cobrado un total de 0.445$, esto no corresponde al precio de las pruebas mostradas en este capítulo, ni al precio total de las pruebas que hemos tenido que hacer para la aplicación. Si no que corresponde a un mes en el que la aplicación no estaba finalizada y se realizan pruebas para depurarla, lo que conllevaba peticiones a todas las APIs, incluida la de AWS. Se puede comprobar cómo hay muchas más peticiones correspondientes a detectar el lenguaje que de sentimientos, esto quiere decir que en ese momento estuvimos probando más este servicio. En total tenemos 3.690.000 unidades (una unidad = 100 caracteres, con cobro mínimo de 3 unidades) entre las dos peticiones necesarias para los dos análisis en el mes de agosto. Lo que nos arroja que cada unidad nos sale a un precio inferior al céntimo de euro.

Este ejemplo nos muestra como por menos de un euro al mes podemos realizar pruebas con esta tecnología sin preocuparnos por el coste. El coste total que ha tenido este servicio en nuestro proyecto es de 0.61 $, como mostramos a continuación:

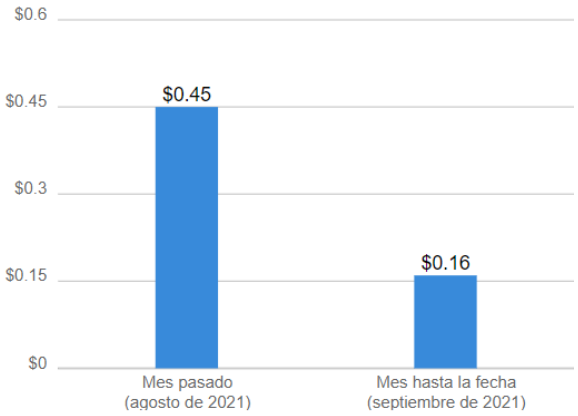


Ilustración 66. Coste de AWS mes de agosto y septiembre del 2021.

Esto se corresponde a las pruebas, tanto para depurar el código como para probar los servicios, que se han realizado a partir del fin del periodo de prueba de doce meses.

5.3 Conclusión

Como conclusión podemos decir que ambos servicios tienen un comportamiento similar, si bien AWS parece "comprender" mejor el contexto de la oración que Azure, acercándose más a lo que podría hacer un ser humano. Esta cualidad de AWS hace que la estimación de la puntuación que hacemos en función a los sentimientos que obtenemos tras el análisis sea más cercana a la media que hacemos de las puntuaciones que dejan los autores de los comentarios en Google Maps.

Con esto hemos finalizado el capítulo de pruebas de este trabajo, en el siguiente capítulo expondremos las conclusiones y las líneas futuras del mismo.

6 CONCLUSIONES

En este último capítulo vamos a mostrar las conclusiones y las líneas futuras de nuestro trabajo.

Para poder exponer de forma clara nuestras conclusiones, las dividiremos en puntos, procediendo de forma similar a como lo hicimos con nuestros objetivos. Relacionaremos las conclusiones con los objetivos.

Respecto al primer punto, el estado del arte:

- En relación con los servicios de *machine learning*, hemos visto que existen multitud de ofertas, muy parecidas entre ellas y con diferentes formas para intentar atraer a los nuevos usuarios a su ecosistema. Tras finalizar este trabajo podemos decir que la elección dependerá del caso concreto donde tengamos que aplicarlo. Ya que, mirando la documentación, y tras comparar, los servicios de AWS y de Azure vemos que poseen métodos muy similares, y la elección de uno u otro no va a depender tanto de lo que nos ofrecen. Dependerán más bien de lo familiarizado que estemos con uno u otro o, con los costes totales que calculemos que van a tener. Si bien es cierto que hemos comprobado que AWS parece comprender mejor el significado completo de una oración, analizando las puntuaciones que nos envían uno y otro servicio vemos que no difieren demasiado, por lo que no consideramos que sea un elemento determinante. Lo que si es cierto es que para los estudiantes Microsoft tiene pruebas y uso gratuito en muchos de sus productos por lo que es una ventaja para tener en cuenta si queremos combinar esta tecnología con otras dentro del ecosistema de Microsoft.
- En cuanto a la elección de los servicios de mapas hemos visto que la mejor opción es la de Google, esto es así debido a la enorme cantidad de usuarios activos que posee y a todas las posibilidades que nos ofrece como desarrolladores su API. Debido a que al ser una API tan popular nos da muchas opciones a la hora de elegir el dispositivo en el que realizar la implementación, sea un dispositivo móvil como teléfono inteligente o en un ordenador.

Respecto al escenario:

- Observamos que la implementación de servicio de Machine Learning empleando la computación en la nube es muy viable. Esto se debe al bajo coste que conlleva, incluso a nivel de pruebas, sin fin comercial. Además, gracias a herramientas como Boto3 o AWS CLI la implementación en un ordenador que soporte Python es inmediata, instalando las librerías y el cliente como se ha explicado en el capítulo tres el ordenador es capaz de ejecutar sin ningún problema los servicios de AWS. Los de Google Maps, al tratarse de peticiones a una API REST, no requieren instalación alguna se software adicional por lo que será soportado siempre y cuando contemos con conexión y con librerías básicas para el manejo de peticiones HTTP. Este último es un requisito obvio e imprescindible para emplear cualquier servicio remoto como es la computación en la nube. En el caso de Microsoft Azure es similar al de AWS, posee librerías que permite usar sus productos en diferentes lenguajes, entre los que se encuentra Python, por lo que también es muy rápido de implementar y de usar.

Respecto a nuestra prueba de concepto:

- Tras finalizar este trabajo hemos visto como es factible pasar de comentarios, escritos en cualquier idioma, en una red social como es Google Maps a datos ordenados. Extrayendo los datos de forma automática y analizándolos mediante *machine learning* sin necesidad de entrenar ninguna máquina y pagando únicamente por la potencia y los servicios que consumimos. Esto supone una gran ventaja ya que nos permite usar los servicios y métodos disponibles cuando nos sea necesario, evitándonos grandes inversiones y permitiéndonos hacer las pruebas o los análisis únicamente cuando lo necesitamos.

6.1 Líneas Futuras

Para finalizar, exponemos algunas posibles líneas futuras para este proyecto:

- Emplear un número mayor de comentarios para cada local. Aumentado la muestra de datos nuestro análisis será más fiable y los resultados más significativos.
- Ampliar el análisis empleando más métodos de AWS Comprehend y de Text Analytics. Por ejemplo, poseen funciones para buscar palabras claves. De esta forma podríamos mejorar el análisis buscando palabras que sean relevantes en el sector en el que nos encontremos, en nuestro caso la hostelería.
- Creación de una interfaz amigable para personas no técnicas. Sería interesante el desarrollo de una interfaz, de forma que un usuario que no esté familiarizado con entornos de desarrollo se sienta más cómodo usando estás herramientas. De manera que acercaríamos esta tecnología a un público aún mayor.
- Probar con otras fuentes y tipo de datos. Por ejemplo, sería interesante analizar tweet sobre diferentes temas o campañas de forma que tengamos una aplicación que emplee los servicios de *Machine learning* en la nube para analizar temas basado en campañas que sean tendencia en un momento determinado. De esta forma podremos probar los servicios con texto en otro formato, tweet en este ejemplo. Además de que serviría para estudiar el impacto que esté teniendo la campaña.
- Implementar el análisis con Google Cloud para comprobar cómo se comporta.

REFERENCIAS

[1] A. Gonzales, «cleverdata,» [En línea]. Available: https://cleverdata.io/conceptos-basicos-machine-learning/. [Último acceso: 15 Julio 2020].

[2] L. J. Aguilar, «COMPUTACIÓN EN LA NUBE: Notas para una estrategia española en cloud computing,» *RevistaIEEE,* nº nº00, nov 2018.

[3] Amazone Web Services, «2020, Amazon Web Services, Inc.,» [En línea]. Available: https://aws.amazon.com/es/free/?nc2=h_ql_pr_ft&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=tier%23always-free%7Ctier%2312monthsfree&awsf.Free%20Tier%20Categories=categories%23ai-ml. [Último acceso: 27 09 2020].

[4] «Amazon Web Services,» 2020. [En línea]. Available: https://aws.amazon.com/es/free/?nc2=h_ql_pr_ft&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Categories=categories%23ai-ml&awsf.Free%20Tier%20Types=tier%2312monthsfree%7Ctier%23always-free. [Último acceso: 15 04 2020].

[5] Microsoft Azure, «Microsoft Azure,» 2020. [En línea]. Available: https://azure.microsoft.com/es-es/pricing/. [Último acceso: 01 Mayo 2020].

[6] «Google Cloud,» 2020. [En línea]. Available: https://cloud.google.com/products/ai/. [Último acceso: 03 05 2020].

[7] DCM, «BPS (Business Publications Spain),» [En línea]. Available: https://www.datacentermarket.es/tendencias-tic/noticias/1123667032809/aws-azure-google-cloud-y-alibaba-ostentan-67-de-cuota-de-mercado-global.1.html. [Último acceso: 04 05 2021].

[8] «Google Maps Platform,» [En línea]. Available: https://developers.google.com/places/web-service/intro?hl=es. [Último acceso: 29 06 2020].

[9] «Google Cloud,» [En línea]. Available: https://cloud.google.com/maps-platform/pricing/?hl=es. [Último acceso: 29 06 2020].

[10] Amazon Web Services, «Boto3 Docs 1.14.61 documentation,» [En línea]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/comprehend.html#Comprehend.Client.batch_detect_dominant_language. [Último acceso: 27 09 20].

[11] Amazon Web Services, «Boto3 Docs 1.14.61 documentation,» [En línea]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/comprehend.html#Comprehend.Client.batch_detect_sentiment. [Último acceso: 27 09 2020].

[1 Amazon Web Services, «Amazon Comprehend,» [En línea]. Available:

2] https://docs.aws.amazon.com/comprehend/latest/dg/how-languages.html. [Último acceso: 27 09 2020].

[1 AWS, «Comience (AWS SDK para Python),» [En línea]. Available:
3] https://docs.aws.amazon.com/es_es/frauddetector/latest/ug/getting-started-python.html. [Último acceso: 06 05 2021].

[1 aws.amazon.com, «AWS,» [En línea]. Available: https://aws.amazon.com/es/sdk-for-python/. [Último
4] acceso: 10 09 2020].

[1 AWS, «Interfaz de línea de comandos de AWS,» [En línea]. Available: https://aws.amazon.com/es/cli/.
5] [Último acceso: 06 05 2021].

[1 Autor, «Este es el ejemplo de una cita,» *Tesis Doctoral,* vol. 2, nº 13, 2012.
6]

[1 O. Autor, «Otra cita distinta,» *revista,* p. 12, 2001.
7]

[1 A. González, «claverdata,» [En línea]. Available: https://cleverdata.io/conceptos-basicos-machine-
8] learning/. [Último acceso: 15 Julio 2020].

[1 aws.amazon.com, «AWS,» [En línea]. Available: https://docs.aws.amazon.com/cli/latest/userguide/cli-
9] chap-welcome.html. [Último acceso: 10 09 2020].

[2 A. W. Services, «Boto3 Docs 1.14.58 documentation,» [En línea]. Available:
0] https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/comprehend.html#Comprehend.Client.batch_detect_sentiment. [Último acceso: 10 09 2020].

APÉNDICE

En esta sección dejamos el código que hemos desarrollado para la prueba del escenario.

main.py:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 18 10:45:12 2021

@author: Antonio Manuel Romera Rodriguez

En este fichero tendremos la función main de nuestra aplicación. Cogeremos
los
números de teléfono de un fichero externos y realizaremos las peticiones para
esos números.

"""
#import de librerías


from google import google
from visionado_html import visionado_html
#from analisisAWS import analisisAWS

#Primero vamos a leer el fichero en el que tenemos los números de teléfono
que
#queremos analizar.

numFile=open('numTelefonos.txt','r')
apiKey=""

fAWS=open('OpinionAWS.txt','w')
fAzure=open('OpinionAzure.txt','w')
"""
*Mediante un bucle for leeremos los números y realizaremos el análisis
"""
indice=0
diccionarioFinal={}
for entity in numFile.readlines():
    #print("el numero de telefono que se nos pasa es: "+entity)
    analisisFinal = google(fAWS,fAzure,numTelefono=entity)
    diccionarioFinal[entity]=(analisisFinal)
    indice+=1
    #Aqui ahora mismo cada numero de teléfono tiene los dos analisis
fAWS.close()
fAzure.close()
#Prueba para ver que recibimos


resultado=visionado_html(diccionarioFinal)
if resultado!=1:
    print("Ha ocurrido un error al generar el HTML")
else:
    print("HTML generado con exito. Fin del Análisis")
```

google.py:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 18 11:10:53 2021

@author: Antonio Manuel Romera Rodriguez

En este fichero se realizarán las peticiones a Maps para poder obtener los
comentarios
de los locales según su número de teléfono.
Recibe como parámetro el numero de telefono del local que queremos analizar
Tras solicitar los comentarios se realizaran llamadas a analisisAWS y
analisisAzure
Ambas devolverán una lista con su analisis estas listas se agruparan en otra
lis
llamada analisis que será lo que devuelva la función.
"""
#imports
import requests
from analisisAWS import analisisAWS
from analisisAZURE import analisisAzure
def google(fAWS,fAzure,numTelefono):

    mykey="AIzaSyCTDYFU4Lju-OLLrKIdje8ApbZTrayG4TM"#Esto sería la clave de
google

url1="https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input
=%2B34"+numTelefono+"&inputtype=phonenumber&fields=place_id&key="+mykey

    response1 = requests.get(url1) #Esta es la forma habitual de realizar una
petición get en Python.

    res1=response1.json()

    apiKey=""
    #Ahora debemos de guardar el placeId para poder hacer luego la petición
de los comentarios.
    for entity in res1["candidates"]:
        entityId1=entity["place_id"]
        apiKey=entityId1

    """
    *Una vez hemos realizado lo anterior procedemos a realizar una
    *segunda llamada a con el objetivo de obtener comentarios de los
    *clientes del local en cuestión, esta vez empleamos el id_place
    *para obtener dicha información.
    """

url12="https://maps.googleapis.com/maps/api/place/details/json?place_id="+api
Key+"&fields=review&key="+mykey
    response12 = requests.get(url12) #Esta es la forma habitual de realizar
una petición get en Python.
    rev_bruto=response12.json()      #Aqui tenemos los comentarios tal cual
nos llegan
```

```
    if len(rev_bruto) ==0:
        print("Error, este local no tiene comentarios, num telefono=
"+numTelefono)

    #Llamamos a los métodos de analisis.
    listaResultadoAWS=analisisAWS(numTelefono,rev_bruto,fAWS)
    listaResultadoAzure=analisisAzure(numTelefono,rev_bruto,fAzure)

    analisis={'AWS':listaResultadoAWS,'AZURE':listaResultadoAzure}
    print(analisis)
    return analisis
```

analisisAWS.py:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 18 13:41:56 2021

@author: Antonio Manuel Romera Rodriguez
En este fichero haremos las llamadas a AWS Comprehend de forma que recibirá
como
parámetro:
*indice: Variable que almacena el número de teléfono del local que
analizamos.
*listaComentarios: Una lista con los comentarios de un local.
*f: Descriptor de un fichero de texto donde guardaremos datos del análisis
Estos comentarios se separarán según sean en inglés o en castellano y se
analizaran
los sentimientos que encierra, Positive, Negative, Mixed. Neutral.
Se devolverá una lista con los valores obtenidos para poder representar los
datos
"""

import boto3
from grafica import grafica
import numpy as np

def analisisAWS(indice,listaComentarios,f):

    #Definimos variables
    positive=0
    negative=0
    mixed=0
    neutral=0
    listaSentimineto=[]
    # el orden de guardado es [positiive,negative,mixed,neutral]
    listaIdioma=[]

    listaIngles=[]
    listaCastellano=[]
    usuPunt=[]
    numComentariosIngles=0
    numComentariosCastellano=0
    client=boto3.client('comprehend')
    for entity in listaComentarios["result"]["reviews"]:
        usuPunt.append(entity["rating"])
```

```
responseIdioma=client.detect_dominant_language(Text=str(entity["text"]))
        aux2=responseIdioma.get('Languages')
        if aux2[0]['LanguageCode']=='en' :
            listaIngles.append(entity["text"])
            numComentariosIngles+=1
        else:
            if aux2[0]['LanguageCode']=='es' :
                listaCastellano.append(str(entity["text"]))
                numComentariosCastellano+=1
            else:
                print('El comentario no esta ni en ingles ni en castellano')

    #Devolvemos una lista con el numero de comentarios en inglés y en
castellano
    listaResultadoIdioma=[] #Primero los comentarios en inglés, luego en
castellano
    listaResultadoIdioma.append(numComentariosIngles)
    listaResultadoIdioma.append(numComentariosCastellano)

    #Comprobamos que hay comentarios en alguno de los dos idiomas
    if numComentariosCastellano!=0 or numComentariosIngles !=0:
        #Primero analizamos los comentarios en ingles
        for entity in listaIngles :
            responseSentimentIngles = client.detect_sentiment(Text=entity,
                                                    LanguageCode='en')
            f.write("Numero de teléfono = "+str(indice)+"\n")
            f.write("\n El comentario analizado es:"+entity+"\n")
            f.write("\n Document Sentiment by AWS in English:
{}".format(responseSentimentIngles["Sentiment"]))
            f.write("\n")
            f.write("Overall scores:\n positive={0:.6f}\n neutral={1:.6f}\n
negative={2:.6f}\n mixed={3:.6f} \n".format(
                responseSentimentIngles["SentimentScore"]["Positive"],
                responseSentimentIngles["SentimentScore"]["Neutral"],
                responseSentimentIngles["SentimentScore"]["Negative"],
                responseSentimentIngles["SentimentScore"]["Mixed"],
            ))
            f.write("\n")
            if responseSentimentIngles["Sentiment"] == 'POSITIVE' :
                positive=positive+1
            else:
                if responseSentimentIngles["Sentiment"] == 'NEGATIVE' :
                    negative=negative+1
                else:
                    if responseSentimentIngles["Sentiment"] == 'MIXED' :
                        mixed+=1
                    else:
                        if responseSentimentIngles["Sentiment"] == 'NEUTRAL'
:
                            neutral=neutral+1
        #Hacemos lo mismo para los comentarios en Castellano.
        for entity in listaCastellano :
            responseSentimentCastellano =
client.detect_sentiment(Text=entity,

LanguageCode='es')
```

```
            f.write("Número de teléfono = "+str(indice)+"\n")
            f.write("\n El comentario analizado es:"+entity+"\n")
            f.write("\n Document Sentiment by AWS in Spanish::
{}".format(responseSentimentIngles["Sentiment"]))
            f.write("\n")
            f.write("Overall scores:\n positive={0:.6f}\n neutral={1:.6f}\n
negative={2:.6f}\n mixed={3:.6f} \n".format(
                responseSentimentIngles["SentimentScore"]["Positive"],
                responseSentimentIngles["SentimentScore"]["Neutral"],
                responseSentimentIngles["SentimentScore"]["Negative"],
                responseSentimentIngles["SentimentScore"]["Mixed"],
            ))
            f.write("\n")
            if responseSentimentCastellano["Sentiment"] == 'POSITIVE' :
                positive=positive+1
            else:
                if responseSentimentCastellano["Sentiment"] == 'NEGATIVE' :
                    negative=negative+1
                else:
                    if responseSentimentCastellano["Sentiment"] == 'MIXED' :
                        mixed=mixed+1
                    else:
                        if responseSentimentCastellano["Sentiment"] ==
'NEUTRAL' :
                            neutral=neutral+1
            #Ya tendríamos los dos análisis hechos. Ahora tocaría agruparlo
en un dic

        f.write("\n")

        listaIdioma.append(numComentariosIngles)
        listaIdioma.append(numComentariosCastellano)
        listaSentimineto.append(positive)
        listaSentimineto.append(negative)
        listaSentimineto.append(mixed)
        listaSentimineto.append(neutral)
        puntMediaUsu=np.mean(usuPunt)
        #Ya tenemos la lista que queremos acoplar al diccionario final.
        analisisCompleto={'AnalisisIdioma':[],
                          'AnalisisSentimiento':[],
                          'raitingUsuMean':0,
                          'numTotalComentarios':0,
                          'ErrorGraficas':0,
                          'PuntMedia':0
                          }
        analisisCompleto['AnalisisIdioma'].append(listaIdioma)
        analisisCompleto['AnalisisSentimiento'].append(listaSentimineto)
        analisisCompleto['raitingUsuMean']=puntMediaUsu
        numTotalComentarios=numComentariosIngles+numComentariosCastellano
        analisisCompleto['numTotalComentarios']=numTotalComentarios

        puntPositive=5*analisisCompleto['AnalisisSentimiento'][0][0]
        puntNegative=1*analisisCompleto['AnalisisSentimiento'][0][1]
        puntMixed=3.5*analisisCompleto['AnalisisSentimiento'][0][2]
        puntNeutral=2.5*analisisCompleto['AnalisisSentimiento'][0][3]

        #Ya tenemos asignada las puntuaciones. La puntuación media del local
será
```

```
        #La suma de todas divido entre el número total de comentarios
        puntMedia=(puntPositive-puntNegative+puntNeutral+
                   puntMixed)/analisisCompleto['numTotalComentarios']
        if puntMedia<0: #Si solo existen comentarios negativos capamos a
cero.
            puntMedia=0
        analisisCompleto['PuntMedia']=puntMedia
        if grafica(indice,analisisCompleto,1)!=0:
            print("Ha ocurrido un error al generar la gráfica\n")
            analisisCompleto['ErrorGraficas']=1

        return analisisCompleto

    else:
        print("Error no hay comentarios para el teléfono: "+str(indice))
        listaIdioma.append("Error")
        listaSentimineto.append("Error")
                #Ya tenemos la lista que queremos acoplar al diccionario
final.
        analisisCompleto={'AnalisisIdioma':[],
                          'AnalisisSentimiento':[]
                          }
        analisisCompleto['AnalisisIdioma'].append(listaIdioma)
        analisisCompleto['AnalisisSentimiento'].append(listaSentimineto)

        if grafica(indice,analisisCompleto,1)!=0:
            print("Ha ocurrido un error al generar la gráfica\n")
            analisisCompleto['ErrorGraficas']=1
        return analisisCompleto
```

clienteAzure.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 10 17:38:40 2021

@author: Microsoft. Sitio Web:
https://docs.microsoft.com/es-es/azure/cognitive-services/text-
analytics/quickstarts/client-libraries-rest-api?tabs=version-3-
1&pivots=programming-language-python#language-detection
Este fichero contiene el método para identificarnos en Microsoft Azure
"""

# use this code if you're using SDK version is 5.0.0
from azure.ai.textanalytics import TextAnalyticsClient
from azure.core.credentials import AzureKeyCredential
key="e4e5fe7eedac4a1484806b67079d435f"
endpoint="https://tfgtest.cognitiveservices.azure.com/"

def authenticate_client():
    ta_credential = AzureKeyCredential(key)
    text_analytics_client = TextAnalyticsClient(
            endpoint=endpoint,
            credential=ta_credential)
    return text_analytics_client
```

analisisAZURE.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 18 13:41:56 2021

@author: Antonio Manuel Romera Rodriguez
En este fichero haremos las llamadas a Azure text analytics para analizar
comentarios
de forma que recibirá como parámetros:
*indice: Variable que almacena el número de teléfono del local que
analizamos.
*listaComentarios: Una lista con los comentarios de un local.
*f:Descriptor de un fichero de texto donde guardaremos datos del análisis
Estos comentarios se separarán según sean en inglés o en castellano y se
analizaran
los sentimientos que encierra, Positive, Negative, Mixed. Neutral.
Se devolverá una lista con los valores obtenidos para poder representar los
datos
"""
from azure.ai.textanalytics import TextAnalyticsClient
from clienteAzure import authenticate_client #Esto es lo que empleamos para
iniciar sesión en Azure
from grafica import grafica


def analisisAzure(indice,listaComentarios,f):

    #Primero realizamos la autenticación
    client = authenticate_client()

    #Definimos variables
    positive=0
    negative=0
    neutral=0
    mixed = 0
    listaSentimineto=[]
    # el orden de guardado es [positiive,negative,mixed,neutral]
    listaIdioma=[]
    usuPunt=[]
    listaIngles=[]
    listaCastellano=[]
    numComentariosIngles=0
    numComentariosCastellano=0
    for entity in listaComentarios["result"]["reviews"]:
       # documents=entity["text"]
        documents=[]
        usuPunt.append(entity["rating"])
        documents.append(entity["text"])
        responseIdioma=client.detect_language(documents = documents,
country_hint = '')[0]
        aux2=responseIdioma.primary_language.name

        if aux2=='English' :
            listaIngles.append(entity["text"])
            numComentariosIngles+=1
        else:
```

```
            if aux2=='Spanish' :
                listaCastellano.append(str(entity["text"]))
                numComentariosCastellano+=1
            else:
                print('El comentario no esta ni en ingles ni en castellano')

    #Devolvemos una lista con el numero de comentarios en ingles y en
castellano
    listaResultadoIdioma=[] #Primero los comentarios en ingles, luego en
castellano
    listaResultadoIdioma.append(numComentariosIngles)
    listaResultadoIdioma.append(numComentariosCastellano)

    #Comprobamos que hay comentarios en alguno de los dos idiomas
    if numComentariosCastellano!=0 or numComentariosIngles !=0:
        #Primero analizamos los comentarios en ingles
        for entity in listaIngles :
            documents=[]
            documents.append(entity)
            responseSentimentIngles =client.analyze_sentiment(documents =
documents)[0]
            f.write("Numero de teléfono = "+str(indice)+"\n")
            f.write("\n El comentario analizado es:"+entity)
            f.write("\n Document Sentiment by AZURE in English: {}"
                    .format(responseSentimentIngles.sentiment)+"\n")
            f.write("Sentence
score:\nPositive={0:.6f}\nNeutral={1:.6f}\nNegative={2:.6f}\n".format(
               responseSentimentIngles.confidence_scores.positive,
               responseSentimentIngles.confidence_scores.neutral,
               responseSentimentIngles.confidence_scores.negative,

            ))
            f.write("\n")
            if responseSentimentIngles.sentiment == 'positive' :
                positive=positive+1
            else:
                if responseSentimentIngles.sentiment == 'negative' :
                    negative=negative+1
                else:
                    if responseSentimentIngles.sentiment == 'neutral' :
                            neutral=neutral+1
                    else:
                        if responseSentimentIngles.sentiment == 'mixed' :
                            mixed=mixed+1

        #Hacemos lo mismo para los comentarios en Castellano.
        for entity in listaCastellano :
            documents=[]
            documents.append(entity)
            responseSentimentCastellano = client.analyze_sentiment(documents
= documents)[0]
            f.write("Numero de teléfono = "+str(indice)+"\n")
            f.write("\n El comentario analizado es:"+entity)
            f.write("\n Document Sentiment by Azure in Spanish: {}"
                    .format(responseSentimentCastellano.sentiment)+"\n")
            f.write("Sentence
score:\nPositive={0:.6f}\nNeutral={1:.6f}\nNegative={2:.6f}\n".format(
```

```
                responseSentimentCastellano.confidence_scores.positive,
                responseSentimentCastellano.confidence_scores.neutral,
                responseSentimentCastellano.confidence_scores.negative,
            ))
            f.write("\n")
            if responseSentimentCastellano.sentiment == 'positive' :
                positive=positive+1
            else:
                if responseSentimentCastellano.sentiment == 'negative' :
                    negative=negative+1
                else:
                    if responseSentimentCastellano.sentiment == 'neutral' :
                        neutral=neutral+1
                    else:
                        if responseSentimentIngles.sentiment == 'mixed' :
                            mixed=mixed+1

            #Ya tendríamos los dos análisis hechos. Ahora tocaría agruparlo
en un dic
        f.write("\n")

        listaIdioma.append(numComentariosIngles)
        listaIdioma.append(numComentariosCastellano)
        listaSentimineto.append(positive)
        listaSentimineto.append(negative)
        listaSentimineto.append(mixed)
        listaSentimineto.append(neutral)
        #Ya tenemos la lista que queremos acoplar al diccionario final.
        analisisCompleto={'AnalisisIdioma':[],
                          'AnalisisSentimiento':[],
                          'raitingUsu':[],
                          'numTotalComentarios':0,
                          'ErrorGraficas':0,
                          'PuntMedia':0
                          }
        analisisCompleto['AnalisisIdioma'].append(listaIdioma)
        analisisCompleto['AnalisisSentimiento'].append(listaSentimineto)
        analisisCompleto['raitingUsu'].append(usuPunt)
        numTotalComentarios=numComentariosIngles+numComentariosCastellano
        analisisCompleto['numTotalComentarios']=numTotalComentarios

        puntPositive=5*analisisCompleto['AnalisisSentimiento'][0][0]
        puntNegative=1*analisisCompleto['AnalisisSentimiento'][0][1]
        puntMixed=3.5*analisisCompleto['AnalisisSentimiento'][0][2]
        puntNeutral=2.5*analisisCompleto['AnalisisSentimiento'][0][3]

        #Ya tenemos asignada las puntuaciones. La puntuación media del local
será
        #La suma de todas divido entre el número total de comentarios
        puntMedia=(puntPositive-puntNegative+puntNeutral+puntMixed
                   )/analisisCompleto['numTotalComentarios']
        if puntMedia<0: #Si solo existen comentarios negativos capamos a
cero.
            puntMedia=0
        analisisCompleto['PuntMedia']=puntMedia
        if grafica(indice,analisisCompleto,0)!=0:
            print("Ha ocurrido un error al generar la gráfica\n")
            analisisCompleto['ErrorGraficas']=1
```

```
        return analisisCompleto

    else:
        print("Error no hay comentarios para el teléfono: "+str(indice))
        listaIdioma.append("Error")
        listaSentimineto.append("Error")
                #Ya tenemos la lista que queremos acoplar al diccionario
final.
        analisisCompleto={'AnalisisIdioma':[],
                          'AnalisisSentimiento':[]
                          }
        analisisCompleto['AnalisisIdioma'].append(listaIdioma)
        analisisCompleto['AnalisisSentimiento'].append(listaSentimineto)

        if grafica(indice,analisisCompleto,0)!=0:
            print("Ha ocurrido un error al generar la gráfica\n")
            analisisCompleto['ErrorGraficas']=1
        return analisisCompleto
```

grafica.py:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 13 17:38:46 2021

@author: Antonio

En este fichero vamos a crear la gráfica correspondiente a un local.
Para ello emplearemos los datos que nos pasa analisisAWS.py y
analisisAzure.py
Parámetros que recibe:
*indice: variable que aloja el número de teléfono del local que analizamos
*aws: Si vale uno la llamada se realiza desde AWS, en caso contrario es desde
Azure.
Esta grafica se usará en el diseño HTML para presentar los datos
Esta función devuelve 1 si todo ha ido correctamente y cero en caso contrario
"""
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

def grafica(indice,analisisCompleto,aws):
    #Lo primero será preparar los datos que recibimos para graficarlos
    if(aws==1):
        #Primero vamos a crear la gráfica de los idiomas
        idiomas=['Ingles', 'Castellano', 'Otro']
        #Definimos la lista de hablantes por idioma
        resta=analisisCompleto['numTotalComentarios']-
(analisisCompleto['AnalisisIdioma'][0][0]+analisisCompleto['AnalisisIdioma'][
0][1])#serían los comentarios que no están ni en ingles ni en castellano

numHablantes=[analisisCompleto['AnalisisIdioma'][0][0],analisisCompleto['Anal
isisIdioma'][0][1],resta]

       #Creamos la gráfica
        fig, ax = plt.subplots()
```

```
        #Colocamos una etiqueta en el eje Y
        ax.set_ylabel('Numero de comentarios')
        #Colocamos una etiqueta en el eje X
        ax.set_title('Numero de Comentarios por Idioma obtenido mediante
AWS')
        #Creamos la grafica de barras utilizando 'idiomas' como eje X y
'numHablantes' como eje y.
        plt.bar(idiomas, numHablantes)
        indice=indice.strip('\n')
        plt.savefig('grafica_Local_AWS_'+str(indice)+'_numHablantes.png')
        #Finalmente mostramos la grafica con el método show()
        plt.show()

        #Ya tendríamos la primera gráfica creada. Ahora creamos la de
sentimientos

        #Lo primero será asignar un valor numérico a cada sentimiento de
forma que podamos hacer una media entre ellos.
        '''
        Positive=5
        Negative=-1
        Mixed=3.5
        Neutral=2.5
        Esta puntuación es arbitraria puesta así por el gráfico de la página:
            https://aws.amazon.com/es/blogs/aws-spanish/analizando-las-
opiniones-de-nuestros-clientes-con-amazon-comprehend/

        '''

        #Ahora vamos a sacar la gráfica de Sentimientos, seguimos los pasos
de antes
        #Creamos la gráfica
        sentimientos=["POSITIVE","NEGATIVE","MIXED","NEUTRAL"]

        positivo=analisisCompleto['AnalisisSentimiento'][0][0]
        negativo=analisisCompleto['AnalisisSentimiento'][0][1]
        mixed=analisisCompleto['AnalisisSentimiento'][0][2]
        neutral=analisisCompleto['AnalisisSentimiento'][0][3]
        usuSentiment=np.array([positivo,negativo,mixed,neutral])
        fig, ax = plt.subplots()
        #Colocamos una etiqueta en el eje Y
        ax.set_ylabel('Numero de comentarios')
        #Colocamos una etiqueta en el eje X
        ax.set_title('Numero de Comentarios por sentimiento obtenido mediante
AWS')

        plt.bar(sentimientos, usuSentiment)
        plt.savefig('grafica_Local_AWS_'+str(indice)+'_Sentimientos.png')
        #Finalmente mostramos la gráfica con el método show()
        plt.show()

    #Parte de Azure
    else:
        #Primero vamos a crear la gráfica de los idiomas
        idiomas=['Ingles', 'Castellano', 'Otro']
        #Definimos la lista de hablantes por idioma
        resta=analisisCompleto['numTotalComentarios']-
(analisisCompleto['AnalisisIdioma'][0][0]+
```

```
analisisCompleto['AnalisisIdioma'][0][1])#serían los comentarios que no están
ni en ingles ni en castellano

numHablantes=[analisisCompleto['AnalisisIdioma'][0][0],analisisCompleto['Anal
isisIdioma'][0][1],resta]

       #Creamos la gráfica
        fig, ax = plt.subplots()
        #Colocamos una etiqueta en el eje Y
        ax.set_ylabel('Numero de comentarios')
        #Colocamos una etiqueta en el eje X
        ax.set_title('Numero de Comentarios por Idioma obtenido mediante
Azure')
        indice=indice.strip('\n')
        #Creamos la grafica de barras utilizando 'idiomas' como eje X y
'numHablantes' como eje y.
        plt.bar(idiomas, numHablantes)
        plt.savefig('grafica_Local_Azure_'+str(indice)+'_numHablantes.png')
        #Finalmente mostramos la grafica con el método show()
        plt.show()

        #Ya tendríamos la primera gráfica creada. Ahora creamos la de
sentimientos

        #Lo primero será asignar un valor numérico a cada sentimiento de
forma que podamos hacer una media entre ellos.
        '''
        Positive=5
        Negative=-1
        Mixed=3.5
        Neutral=2.5
        '''

        #Ahora vamos a sacar la gráfica de Sentimientos, seguimos los pasos
de antes
        #Creamos la gráfica
        sentimientos=["POSITIVE","NEGATIVE","MIXED","NEUTRAL"]

        positivo=analisisCompleto['AnalisisSentimiento'][0][0]
        negativo=analisisCompleto['AnalisisSentimiento'][0][1]
        mixed=analisisCompleto['AnalisisSentimiento'][0][2]
        neutral=analisisCompleto['AnalisisSentimiento'][0][3]
        usuSentiment=np.array([positivo,negativo,mixed,neutral])
        fig, ax = plt.subplots()
        #Colocamos una etiqueta en el eje Y
        ax.set_ylabel('Numero de comentarios')
        #Colocamos una etiqueta en el eje X
        ax.set_title('Numero de Comentarios por sentimiento obtenido mediante
Azure')

        plt.bar(sentimientos, usuSentiment)
        plt.savefig('grafica_Local_Azure_'+str(indice)+'_Sentimientos.png')
        #Finalmente mostramos la grafica con el método show()
        plt.show()

    return 0
```

visionado_HTML:

```
# -*- coding: utf-8 -*-
"""
Created on Wed May  5 11:52:23 2021
Este método se usa para rellenar la plantilla y mostrar la información
obtenida de los análisis.
Parámetros:
*diccionarioVisionado: Resultado de los análisis. Donde se encuentra toda la
información recabada.
@author: anton
"""
import os
from string import Template

def visionado_html(diccionarioVisionado):

    filein=open('Template/plantilla.html')
    dicaux=[]
    for llave in diccionarioVisionado:
        dicaux.append(llave)

    #Leemos el archivo

    src = Template(filein.read())
    #guardamos los teléfonos en las variables
    print(diccionarioVisionado)


    numeroTelefono1=dicaux[0]
    numeroTelefono2=dicaux[1]
    numeroTelefono3=dicaux[2]


    puntMedia1=str(diccionarioVisionado[numeroTelefono1]['AWS']['PuntMedia'])
    puntMedia2=str(diccionarioVisionado[numeroTelefono2]['AWS']['PuntMedia'])
    puntMedia3=str(diccionarioVisionado[numeroTelefono3]['AWS']['PuntMedia'])


puntUsu1=str(diccionarioVisionado[numeroTelefono1]['AWS']['raitingUsuMean'])

puntUsu2=str(diccionarioVisionado[numeroTelefono2]['AWS']['raitingUsuMean'])

puntUsu3=str(diccionarioVisionado[numeroTelefono3]['AWS']['raitingUsuMean'])


puntMedia1Azure=str(diccionarioVisionado[numeroTelefono1]['AZURE']['PuntMedia
'])

puntMedia2Azure=str(diccionarioVisionado[numeroTelefono2]['AZURE']['PuntMedia
'])

puntMedia3Azure=str(diccionarioVisionado[numeroTelefono3]['AZURE']['PuntMedia
'])
    numeroTelefono1=numeroTelefono1.replace("\n","")
    numeroTelefono2=numeroTelefono2.replace("\n","")
    numeroTelefono3=numeroTelefono3.replace("\n","")
```

```
    #CREAMOS UN DICCIONARIO
    rutaIdioma1="../grafica_Local_AWS_"+numeroTelefono1+"_numHablantes.png"

rutaSentimiento1="../grafica_Local_AWS_"+numeroTelefono1+"_Sentimientos.png"

rutaIdiomaAzure="../grafica_Local_Azure_"+numeroTelefono1+"_numHablantes.png"

rutaSentimientoAzure="../grafica_Local_Azure_"+numeroTelefono1+"_Sentimientos
.png"

    rutaIdioma2="../grafica_Local_AWS_"+numeroTelefono2+"_numHablantes.png"

rutaSentimiento2="../grafica_Local_AWS_"+numeroTelefono2+"_Sentimientos.png"


rutaIdiomaAzure2="../grafica_Local_Azure_"+numeroTelefono2+"_numHablantes.png
"

rutaSentimientoAzure2="../grafica_Local_Azure_"+numeroTelefono1+"_Sentimiento
s.png"

    rutaIdioma3="../grafica_Local_AWS_"+numeroTelefono3+"_numHablantes.png"

rutaSentimiento3="../grafica_Local_AWS_"+numeroTelefono3+"_Sentimientos.png"


rutaIdiomaAzure3="../grafica_Local_Azure_"+numeroTelefono3+"_numHablantes.png
"

rutaSentimientoAzure3="../grafica_Local_Azure_"+numeroTelefono3+"_Sentimiento
s.png"

    d={
'prueba1':numeroTelefono1,'puntUsu1':puntUsu1,'puntMedia1':puntMedia1,
       'rutaIdioma1prueba':rutaIdioma1,
       'rutaSentimiento1':rutaSentimiento1,'puntMediaAzure1':puntMedia1Azure,
       'rutaIdioma1pruebaAzure':rutaIdiomaAzure,

'rutaSentimientos1pruebaAzure':rutaSentimientoAzure,'prueba2':numeroTelefono2
,
       'puntUsu2':puntUsu2,
       'puntMedia2':puntMedia2,'rutaIdioma2':rutaIdioma2,

'rutaSentimiento2':rutaSentimiento2,'puntMediaAzure2':puntMedia2Azure,'rutaId
ioma2pruebaAzure':rutaIdiomaAzure2,

'rutaSentimientos2pruebaAzure':rutaSentimientoAzure2,'prueba3':numeroTelefono
3,'puntUsu3':puntUsu3,
       'puntMedia3':puntMedia3,'rutaIdioma3':rutaIdioma3,

'rutaSentimiento3':rutaSentimiento3,'puntMediaAzure3':puntMedia3Azure,'rutaId
ioma3pruebaAzure':rutaIdiomaAzure3,
       'rutaSentimientos3pruebaAzure':rutaSentimientoAzure3}
    #Sustituimos la variable de Python por la que queremos que aparezca en
HTML
```

```
result=src.substitute(d)
try:
    os.mkdir("perfiles")
    filein2=open('perfiles/analisis.html','w')
    filein2.writelines(result)
    print("Creando carpeta y archivo")
    print("Guardando archivo")

except OSError:
    if os.path.exists("perfiles"):
        filein2=open('perfiles/analisis.html','w')
        filein2.writelines(result)
        print("Guardando archivo")

filein.close()
filein2.close()

return 1
```