



UNIVERSIDAD DE JAÉN
Politécnica Superior de Jaén

Trabajo Fin de Grado

ANÁLISIS DE LA ACTIVIDAD EN TWITTER DURANTE UN EVENTO CONCRETO

Alumno: David Baudet Moreno

Tutor: Prof Manuel García Vega
Dpto: Lenguajes y Sistemas Informáticos

Mayo, 2020



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Manuel García Vega tutor del Proyecto Fin de Carrera titulado: “Análisis de la actividad en Twitter durante un evento concreto” que presenta David Baudet Moreno autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén 13 Julio de 2020

El alumno:

El tutor:

AGRADECIMIENTOS

En primer lugar, me gustaría reconocer el inestimable apoyo de mis padres, que durante el transcurso de esta difícil carrera me han animado y apoyado a seguir estudiando, sin su ayuda no hubiese llegado tan lejos. Por otra parte, quiero agradecer a mis profesores por todo el conocimiento que me han enseñado en todos estos años. Y por último a mi tutor Manuel García Vega, por su guía, por su ayuda y por su paciencia a la hora de acompañarme en la ejecución de este TFG.

Índice

1. Introducción	8
2. Summary	9
3. Objetivos	10
4. Estado del Arte	11
5. PlanificacionTemporal	13
6. Twitter	22
6.0.1. Hashtag	22
6.0.2. Retweet y favorito	22
7. Análisis de herramientas	24
7.1. Entorno de desarrollo	24
7.1.1. ¿Por qué se ha elegido Pycharm?	25
7.2. Software y librerías	26
7.2.1. API de Twitter	26
7.2.2. Tweepy	27
7.3. Equipo hardware	36
8. Captura de datos	38
8.0.1. Tipos de consultas	38
8.0.2. Funciones de la Api	39
8.0.3. Tipo de resultados	40
8.0.4. Tipo de modo de tweets	40
8.0.5. Argumentos de captura	40
8.0.6. Variables de entorno	41
8.0.7. Funciones de captura	41
8.0.8. Estructura de captura	50

8.0.9. Observaciones de los datos capturados	52
8.0.10. Datos capturados	54
9. Análisis Datos	56
9.0.1. Librerías de detección	56
9.0.2. Librerías de agrupamiento	57
9.0.3. Palabra relacionadas con el tema	59
9.0.4. Api Wikipedia	63
9.0.5. Búsqueda de valores en los tweets	64
9.0.6. Análisis Morfosintáctico	66
9.0.7. Bokeh	70
9.0.8. Clases creadas de apoyo	101
9.0.9. Variables de análisis	104
9.0.10. Funciones creadas	106
9.0.11. Estructura de el análisis de datos	117
9.0.12. Experimentos	118
10.Desarrollo de un portal web para pruebas	119
10.0.1. Flask	119
10.0.2. Características más importantes de Flask	120
10.0.3. Lanzar flask	121
10.0.4. Vistas creadas y funciones creadas	124
10.0.5. Plantillas creadas	137
10.0.6. Variables	157
11.Conclusiones	158
12.Conocimientos adquiridos por el alumno	163
13.Trabajos futuros	164
A. Diario de trabajo	168
A.0.1. Búsqueda de información	168
A.0.2. Planificación por etapas	168
B. Instalación	175
B.0.1. Ubuntu	175
B.0.2. Windows	175
B.0.3. Macosx	175
B.0.4. Una vez instalado Pip	176

B.0.5. Parseador Stanford	177
B.0.6. Guía de uso	178
C. Bibliografía	181

Capítulo 1

Introducción

Con este proyecto de análisis de datos en Twitter[1] durante un evento concreto, se trata de implementar un sistema de tratamiento de mensajes cortos que se recupera de dicha aplicación. Se recogerán los tweets más relevantes de un accidente, evento social, un desastre natural o algo que tenga gran importancia. Para realizar esta recogida de información se realizará un etiquetado de los tweets, clasificándolos por temas, usuarios y fuentes de información oficiales; para elaborar un análisis automático de lo seleccionado.

Para la captura de datos, se ejecutarán varias funciones de captura de tweets, el análisis de los hashtag de dicho suceso, de las principales ciudades españolas y en caso de tener relevancia con el tema de estudio, realizar una captura con dicho hashtag.

La captura se efectuará durante el tiempo designado que introduzca el usuario, para aumentar el análisis, se estudiarán distintas formas de encontrar palabras relacionadas con el tema de búsqueda, usando los valores introducidos por el usuario y con el uso de la Api de Wikipedia. Para el análisis se utilizará un algoritmo de aprendizaje automático utilizando la función K-means, un conjunto de gráficas creadas con la biblioteca Bokeh para que el usuario tenga unas gráficas interactivas, además de poder descargar el contenido de las capturas junto a los análisis de los tweets en formato web.

Metodología a desarrollar.

- Estudio de la Api de Twitter.
- Estudio y selección de los algoritmos a utilizar.
- Detección de entidades.
- Automatización de la recuperación de los tweets.
- Integración de todos los módulos.

Capítulo 2

Summary

With this project of data analysis on Twitter during a specific event, the idea is to implement a system for processing short messages that are retrieved from this application. The most relevant tweets from an accident, social event, a natural disaster or something of great importance will be collected. In order to carry out this information collection, the tweets will be labelled, classified by subject, user and official information source; in order to prepare an automatic analysis of what has been selected.

For the data collection, several functions will be executed to capture tweets, the analysis of the hashtag of that event, of the main Spanish cities and in case of having relevance with the topic of study, to carry out a capture with that hashtag.

The capture will be made during the designated time that the user enters, to increase the analysis, different ways of finding words related to the search topic will be studied, using the values entered by the user and with the use of the Api of Wikipedia. For the analysis an automatic learning algorithm will be used using the K-means function, a set of graphs created with the Bokeh library so that the user has interactive graphs, in addition to being able to download the content of the captures together with the analysis of the tweets in web format.

Methodology to be developed.

- Twitter Api Study.
- Study and selection of the algorithms to be used.
- Entity detection.
- Automation of tweet retrieval.
- Integration of all modules.

Capítulo 3

Objetivos

- Implementación de un algoritmo que recupere tweets mediante consulta a la Api.
- Implementación de un algoritmo de agrupamiento no supervisado.
- Detección de entidades.
- Análisis estadístico de los tweets.
- Análisis Morfosintáctico

Capítulo 4

Estado del Arte

En internet hay muchos estudios que realizan análisis de tweets para algún propósito, tanto análisis de sentimientos[2] o estudio de la composición de un trending topic, aquí unos ejemplos de estudios que se apoyan en Twitter para dar resultados:

- Talleres que explican el tema del machine learning utilizando las redes sociales como puede ser el caso: (Workshops: Mi Primera Neurona + Analítica con Redes Sociales[3]) en Colombia, Trabajos de fin de grado de la Universidad Autónoma de Madrid, relacionados con sistemas de búsqueda y análisis basado en Twitter, que buscan la relevancia de tweets por medio de entrenamiento de redes neuronales o preguntando a usuarios.
- Herramientas como Tweet Binder[4] realizan un análisis de los trending topic en Twitter, pero siempre desde el ámbito más social de la aplicación, no en la detección de agrupamientos en el conjunto de tweets, ni análisis de palabras clave sobre un conjunto de tweets.
- Trendinalia[5] realiza un análisis de los trending topic pero sobre el contenido de los tweets, cuantos de ellos son retweet y los usuarios que han escrito en este trending topic.

Dentro de la Universidad de Jaén[6] se encuentra el grupo de investigación SINAI, este lidera la Red de Dinamización de actividades en tecnologías de Procesamiento del Lenguaje Natural (PLN.net[7]). Dicho grupo se constituye por 13 grupos de investigación de diferentes universidades de España, entre los muchos proyectos que han desempeñado estos 13 grupos de investigación cabe destacar:

- Pictogrammar[8], comunicación basada en pictogramas con conocimiento lingüístico.

- COPOS[9]: Corpus Of Patient Opinions in Spanish. Application of Sentiment Analysis Techniques.

El grupo de investigación SINAI[10] además tiene otros proyectos a destacar como son:

- FIRST, haciendo el lenguaje escrito más accesible.
- REDES, descubriendo la identidad digital.
- Análisis de opiniones y sentimientos.
- Yotta, tecnología del lenguaje aplicada a trastornos del lenguaje.

También son los responsables de el TASS[11] taller análisis de sentimientos y seguimiento de opinión en redes sociales, en este realizan estudio sobre el diseño y implementación de sistemas capaces de analizar sentimientos en los usuarios de las redes sociales.

Debido a la gran aportación de la Universidad de Jaén en el campo del procesamiento del lenguaje natural, los apuntes de la asignatura procesamiento del lenguaje natural han sido consultados para la realización de este trabajo de fin de grado.

Capítulo 5

Planificación Temporal

Para la realización de forma correcta del trabajo de fin de grado, se consideró el uso de un modelo de desarrollo de software, es importante que el modelo a elegir tenga la posibilidad de interactuar con el cliente, se presentarán pequeños prototipos del proyecto final al tutor.

Se ha elegido para el desarrollo del proyecto un modelo en espiral, debido a la naturaleza del trabajo de fin de grado que es de carácter teórico-experimental, en este proyecto no se sabe si se va a poder llegar a una solución, por ello se irán analizando los riesgos de la implementación del software.

Beneficios del modelo en espiral:

Es importante esbozar los beneficios de realizar el proyecto con el siguiente modelo:

- Dado que existen riesgos de no conseguir una solución al proyecto por la naturaleza de este en el proyecto, el modelo en espiral se utiliza mucho para este tipo de proyectos.
- Se tiene un feedback con el cliente que en este caso es el tutor del proyecto, se entregaran pequeños prototipos a medida que se vayan integrando nuevas funcionalidades al proyecto. En el caso de que haya conflictos entre el diseño preestablecido y la opinión del cliente al tener un carácter cíclico se pueden comprobar constantemente estos además de modificarse si es necesario.

Inconvenientes de este modelo:

En algunas ocasiones puede que se introduzcan al producto final, algunas de las partes que se han prototipos, se tiene que tener especial cuidado en este aspecto de que el producto final solo tenga prototipos que hayan sido testeados sin errores.

Otro inconveniente que presenta este modelo es el no saber el número de ciclos o iteraciones que se van a realizar, esto conlleva a retrasos en el desarrollo, además de arrastrar posibles errores durante muchas iteraciones debido a que no se detecten.

Los motivos principales por el que se han elegido este modelo son:

- Tiene interacción con el cliente en cada ciclo.
- Al no tener al principio un estudio sobre el problema, ni el número de funciones que se tienen que implementar, sino que se va a implementar funcionalidades a medida que se vayan analizando en cada iteración.

En este modelo se va a considerar en cada iteración los siguientes puntos:

Cada iteración o ciclo presenta los siguientes puntos:

- **Objetivos:**
En cada iteración se discuten los objetivos con el cliente, se plantean posibles alternativas para cumplir cada objetivo.
- **Análisis y evaluación de riesgos:**
Con cada nueva funcionalidad hay que realizar un análisis del impacto que esta tiene sobre el proyecto final que se esta implementando, se buscan alternativas.
- **Desarrollo:**
Se añaden las nuevas funcionalidades, se prueban y se testean.
- **Planificación del siguiente ciclo**
Se buscan errores y se proponen soluciones, o alternativas para solucionar los errores que se hayan encontrado en la iteración, se plantean nuevos objetivos para subsanar los errores o para incluir nuevas funcionalidades al proyecto.

Iteración nº1: Objetivos: Se estableció conversaciones con el tutor del proyecto para encontrar los primeros objetivos, se contempló varios lenguajes de programación para el proyecto pero se decantó por Python, se tendría que realizar un estudio sobre las api de twitter y sobre los wrappers que tiene Python, que límites tiene la Api de Python para el uso de consultas en Twitter. Conseguir usuarios relacionados con la búsqueda de catástrofes. Encontrar una forma de encontrar hashtag, usuarios mencionados en los tweets y encontrar el tweet más antiguo sobre el tema de búsqueda. Análisis y Riesgos:

- Problemas con la Api, límite en la descarga de tweet, tipo de formato de tweet y como adaptarlo a Python.
- Se encontró problemas para la búsqueda de usuarios relacionados con la búsqueda, debido a que el tener una base de datos con los usuarios, haría que la aplicación solo fuese destinada a catástrofes, uno de los objetivos del TFG es el análisis de eventos , se encontrarían problemas a la hora de encontrar usuarios de cada evento de distinto ámbito, esta parte la haría el usuario manualmente en el caso de usar una base de datos.

- Habría que encontrar una forma de reconocer los hashtag y los usuarios mencionados en los tweets, sin que la complejidad del programa hiciese eterna la ejecución.
- Un problema que se puede encontrar es el como saber si un usuario ha escrito algún tweets sobre el tema de búsqueda.
- Problema con la repetición de los tweets de los usuarios, confeccionar estructuras de datos que quiten la repetición de lo tweets.

Desarrollo: Al ser la primera solo fue una fase de análisis con el cliente para plantear los objetivos, tipo de lenguaje de programación. Planificación siguiente iteración

- Se realizará la búsqueda y confeccionamiento de estructuras que eviten la repetición de los tweets.
- Alternativas a la base de datos con usuarios fijos.
- Realizar búsquedas en español de tweets relacionados con la búsqueda del usuario.
- Confeccionar un algoritmo capaz de encontrar el tweet más antiguo sobre el tema de búsqueda general, buscar los elementos que contiene un tweet y cuales de ellos tienen relación la fecha de publicación.
- Estudiar los límites de la Api, así con manejo de excepciones.
- Encontrar un comparador de fechas entre los tweets.

Iteración nº2: Objetivos:

Buscar una forma de encontrar los usuarios que tienen más relevancia con el tema de búsqueda que el usuario introduce. Realizar primera pruebas con la Api descargando el contenido, examinar los argumentos que tiene cada tweets. Buscar un comparador de tweets para encontrar el más antiguo de una captura, encontrar hashtag relacionados con el objeto de búsqueda. Conseguir información sobre los limites de Api.

Análisis y Riesgos:

- Después de realizar un estudio de todos los Wrappers de Python que conectan con la Api de Twitter, se ha decantado por Tweepy.
- Se ha encontrado una forma de conseguir usuarios de forma automática que tenga relevancia con el tema de búsqueda introducido, un problema es que aunque en la implementación se introduzcan parámetros para que solo sean usuarios en español, devuelve los españoles más algunos en inglés que tienen relevancia con el objeto de búsqueda.

- Se ha analizado la estructura de datos de Tweepy confeccionar un Tweet, tiene muchos parámetros que no tienen relevancia con el objeto de estudio, se ha de estudiar que tipo de datos pueden servir para el estudio y cuales no.
- Se ha realizado algunas capturas de datos, se ha encontrado que la Api de Tweepy tiene un límite a la hora de descargar tweets, esto conlleva a la búsqueda de alternativas para ser más eficientes capturando tweets, podría darse el caso de que hubieran tiempos muy largos en los cuales no se está capturando tweets.

Desarrollo:

Se ha implantado una pequeña aplicación que captura datos de Twitter, sin guardarlos solo los muestra, además de encontrar los usuarios más relevantes de la búsqueda.

Planificación siguiente iteración Se debe encontrar una interfaz viable para la ejecución del programa. Encontrar una forma de realizar la captura de datos, estando el menor tiempo esperando.

Iteración nº3:

Objetivos:

Investigar interfaces de escritorio para Python, Realizar de forma de más eficiente la captura de tweets.

Análisis y Riesgos:

- Una de las primeras bibliotecas de que se encuentran en Python para realizar interfaces es Tkinter, se han encontrado problemas con la interfaz porque no es capaz de mostrar los emoticonos de los tweets al no saber como mostrarlos, da un error, se tiene que encontrar una forma de quitar los emoticonos de los tweets para que esto no afecte.
- Se ha investigado en el uso de la Api de Twitter y pueden utilizarse varias cuentas de desarrollador a la vez, esto puede tener un riesgo de que se bloqueen todas las cuentas de la aplicación o Twitter nos intente banear de su uso.

Desarrollo: Se ha desarrollado una aplicación que descarga tweets los muestra en una interfaz hecha con Tkinter, pero tiene problemas a la hora de mostrar los emoticonos.

Planificación siguiente iteración Se ha contactado con el tutor para enseñar los progresos realizados y para enseñar las funcionalidades que se han implementado en el prototipo de la aplicación.

Después de la aplicación ha querido que se cambiara la interface por una interface web, esto conlleva el estudio de las bibliotecas que tiene python para la creación de paginas web.

Se han encontrado diferentes posibilidades a la hora de descargar tweets, en tiempo real, por tweets populares, por tweets recientes y una forma mixta. Se realizarán pruebas con los tres para ver que cantidad de tweets es capaz de descargar, se pensarán en el número de claves de acceso a la Api se requieren para que todas los tipos de funciones se puedan utilizar.

Iteración nº4: Objetivos: Probar cada una de las formas con las que se puede descargar tweets. Crear nuevas cuentas de Twitter para tener la posibilidad a más acceso a más tweets. Buscar entornos web en Python para crear la página web en la que se ejecute la aplicación.

Análisis y Riesgos:

Se han probado las tres formas de descargar tweets en tres funciones distintas, la función que contiene la llamada a tweets populares no devuelve ningún tweets. Se ha introducido diferentes valores de prueba y el resultado es el mismo, para los tweets que son más recientes devuelve los tweets que han sido publicados con la menor brevedad. Aún así seguimos con el problema de que la Api de Twitter retorna tweets aleatorios que satisfagan los parámetros de llamada, esto hace que se repitan muchos tweets. Como riesgo, esto haría que todos los análisis que se hicieran en el futuro tuvieran unos resultados erróneos. Se ha pensado en estructuras de datos con conjuntos. El uso de la llamada a la Api para tweets que son mixtos entre populares y recientes da buenos resultados porque en este tipo de llamada si retornan los tweets que son los más populares. Por último, se utilizó la llamada en tiempo real los tweets que se descargan tiene una antigüedad desde 2 horas a 10 minutos, esto es debido a factores de que siempre no se están publicando tweets que son del tema de búsqueda.

Se ha buscado diferentes bibliotecas para la creación de páginas web, entre estas se han elegido para el análisis Flask y Django. Flask es un microframework y su curva de aprendizaje es mucho menor que la de Django.

Por este motivo se va utilizar Flask para el proyecto, además de que tiene una gran comunidad detrás los que ayudara a futuras dudas.

Desarrollo:

Una aplicación con las descargas de tweets con los diferentes tipo de capturas de tweets.

Planificación siguiente iteración

- Se ha de implementar un sistema que limpie los tweets, ponga en minúsculas y elimine las palabras vacías.
- Se estudiará diferentes librerías para encontrar: localizaciones y organizaciones.
- Estudiar diferentes métodos para encontrar palabras del mismo contexto, dado que la Api de Twitter utiliza como parámetro de búsqueda palabras clave.

Iteración nº5: Objetivos:

- Implementar un método para limpiar tweets.
- Estudiar Api que al pasarle una palabra nos digan si es o no una localización.
- Implementar una función que encuentre las organizaciones de un texto, utilizando nltk.

Análisis y Riesgos:

Se han detectado una serie de Riesgos al empezar a realizar la limpieza de los tweets, esto es debido a que los usuarios de Twitter utilizan muchos emoji para la escritura de sus tweets, además se ha experimentado errores de compilación debidos por que además de escribir en los tweets emoji los utilizan también para campos como localización y nombre del usuario. Debido a esto la implementación del algoritmo se puede ver afectada por tener que limpiar más campos de los tweets que los que se habían propuesto al principio.

Otro problema que se ha detectado es la poca cantidad de tweet que se descargan, se tiene que realizar un estudio sobre que palabras pueden generalizar más el ámbito del estudio del análisis y buscar una forma en la que se pueda encontrar la raíz de las palabras para usarla como palabra clave.

Además se han encontrado problemas en el uso de Ápi de geolocalizaciones debido a los límites que dan sus versiones gratuitas un ejemplo de estas ha sido geonames, debido a estos errores han de buscarse nuevas Api o librerías.

Planificación siguiente iteración Se ha de investigar una forma de quitar todos los emoji de una frase, encontrar una forma de encontrar las raíces de una palabra, buscar diferentes métodos para detectar localizaciones en los textos.

Iteración nº6: Objetivos: Investigar la forma de encontrar palabras que tengan relación con la palabra introducida con el usuario, detectar la raíz de la palabra para su posterior uso en las búsquedas, estudiar Spacy para encontrar localizaciones. Análisis y Riesgos:

Se han buscado diferentes Api o librerías para el uso de sinónimos o palabras relacionadas, se ha llegado a la conclusión que las Api que existen tienen coste al uso por ello se han descartado o tienen un límite de uso. Para el tema de palabras con el mismo contexto se ha investigado el uso de tesauros para la realización de los análisis, todo esto es para solucionar el problema del contexto de los tweets. Los principales tesauros gratuitos que hay o no tienen Api de acceso o los resultados que devuelven son muy pobres, se van a tener que buscar alternativas para la diferenciar los tweets relevantes de los menos relevantes relacionados con el tema de estudio.

Planificación siguiente iteración Se va a estudiar diferentes Api de bibliotecas o enciclopedias, para encontrar textos que tengan palabras relacionadas con el tema de búsqueda, se va a diseñar un sistema de puntuaciones para diferenciar los tweets más relevantes, además de darle la posibilidad al usuario de introducir el mismo unos valores que buscar en los tweets.

Iteración nº7: Objetivos: Investigar la Api de Wikipedia Análisis y Riesgos: La Api de Wikipedia da buenos resultados, pero al utilizarla se encuentran problemas de textos, predefinidos que se devuelven sea cual sea la consulta. La opción de sumario es la mejor que se puede utilizar ahora hay que limpiar el texto y quedarnos con lo más importante. Esta Api contiene el riesgo de que al ser gratuita tiene límite de uso. Desarrollo:

Se ha desarrollado una función que realiza la consulta sobre la Api y luego otra que filtra

el texto para quedarnos con lo más importante.

Planificación siguiente iteración

Investigar algoritmos de agrupamiento no supervisado y ver cual da mejor resultado con el tipo de texto que tiene los tweets.

Iteración nº8: Objetivos: Estudiar algoritmos de agrupamiento y ver cuales tienen mejor resultado con los tweets. Se va a estudiar DBSCAN, MeanShift y KMeans.

Análisis y Riesgos:

Dentro de los tres algoritmos que se han estudiado los mejores resultados han sido los de KMeans, se ha encontrado otro problema que es buscar un método para encontrar un número óptimo de Cluster.

Se han estudiado el método del codo, dendogramas y el método Gap, la elección ha sido el método por facilidad en su uso, basta con calcular la pendiente más grande en una gráfica generada con las distorsiones que genera el KMeans.

Planificación siguiente iteración

Se empezará a estudiar framework webs en Python, para la creación de la web de prueba, se va a estudiar los framework Flask y Django.

Iteración nº9: Objetivos:

Estudiar como realizar un análisis morfosintáctico de los tweets.

Análisis y Riesgos:

Se han analizado distintos métodos para realizar los análisis morfosintácticos, pero hemos encontrado diversos problemas para dar con una solución.

- En la versión de nltk se han encontrado problemas para introducir las variables de forma iterativa en un algoritmo.
- La forma en la que se representa al pasarla a texto no se muestra bien los niveles de análisis.

Se ha investigado una forma de representar los análisis morfosintácticos en forma de imagen y representarlos en la página web en un apartado exclusivo a él, pero el marco que utiliza este tipo de análisis está hecho con Tkinter el cual no es capaz de soportar un gran número de iteraciones en distintas oraciones. Por ello, esta idea de representarse en imágenes se ha descartado por fallar de forma reiterada de forma arbitraria, los problemas que ha dado se han probado en distintos conjuntos de tweets descargados, dándose error distintos tamaños de los conjuntos y en distintos momentos, por estas razones se ha descartado este tipo de representación.

Los análisis morfosintácticos se añadirán al corpus completo que se genera con cada iteración.

Planificación siguiente iteración Se va a estudiar los diferentes framework web más famosos de Python.

Iteración nº10: Objetivos:

- Se va utilizar Flask como framework el motivo es que la curva de aprendizaje es mucho menor que la de Django.
- Se va a estudiar como implementar una página web simple, para mostrar los resultados y el transcurso de la ejecución de la aplicación, la página web estará estructurada en una serie de vistas:
 - Una breve introducción al TFG.
 - Un formulario para la introducción de los valores de la ejecución.
 - Una vista con el transcurso de la captura de tweets. En esta vista se mostrarán los tweets que se van capturando, se añadirán unas etiquetas para mostrar la ejecución del programa para saber la etapa de la ejecución.
 - Se incluirá una vista con los resultados sin los datos capturados solo los análisis y se añadirá unos botones para descargar tanto el archivo generado como los csv de cada parte.

Análisis y Riesgos: Se hará especial hincapié en el que la generación del análisis se cree en un solo archivo, este contendrá la página web junto los estilos de la pagina. Esto se va a realizar para simplificar el número de archivos que tiene que tener el usuario para poder ver el archivo y descargarlo. Uno de los riesgos al realizar este tipo de implementación es que todo el contenido del análisis estará junto, habrá que estudiar una forma de desarrollar un espacio para cada parte del análisis, además del uso de un menú plegable para quitar el menor espacio de pantalla al usuario. Desarrollo:

El desarrollo de una web para mostrar los datos y interactuar con ellos.

Planificación siguiente iteración Se han encontrado una serie de problemas para la implementación de la página web, el problema de la caché del navegador es que al realizar muchas pruebas en el mismo ordenador. El navegador tiende a guardar los diferentes ficheros en su caché esto hace que se al realizar un análisis sobre un tema se descarguen al final los archivos de un experimento pasado.

Se va a estudiar una forma de desactivar la caché para la siguiente iteración y su problema con las imágenes. Las imágenes que se creen durante la ejecución del programa tienen que estar guardadas en la carpeta Static, esto hará que se tengan que tener métodos de limpieza para cada ejecución del programa.

Por último, se va mejorar el menú del archivo descargable esto es por consejo del tutor el cual me ha aconsejado, un menú retráctil, con un submenú con las capturas de datos.

Iteración nº11: Objetivos:

Estudiar menú con Javascript y JQuery.

Análisis y Riesgos:

Se han tenido problemas con la forma de preseleccionar una parte del análisis. Al principio de carga de la web, se han encontrado problemas con tablas de datos por el formato de los tweets ya que estos no tienen saltos de línea y el texto se sale de las tablas, esto se ha corregido con las funciones scroll de las hojas de estilo.

Planificación siguiente iteración Final de la implementación del TFG

Capítulo 6

Twitter

Twitter[1] es una red social creada en marzo de 2006 por Jack Dorsey, esta red social tiene estructura de micro blog.

El usuario puede escribir pequeños textos de 280 caracteres, incluyendo tanto imágenes, gif, vídeos o enlaces.

6.0.1. Hashtag

Al utilizar el carácter # delante de una palabra se convierte en un contenedor de tweets, su nombre es hashtag. Todos los tweets que tienen en común el mismo hashtag, por ejemplo #COVID-19, están dentro de el mismo contenedor de tweets:

- #COVID-19 es la plaga que nos viene a demostrar el fracaso de la Salud Mundial, junto con el impacto de la globalización. Las irresponsabilidades, nos afectan a todos.
- Si eres de Madrid, haz caso a María #quedatencasa #COVID-19

6.0.2. Retweet y favorito

Estos dos términos van a ser muy recurrentes en la documentación de este TFG, en el ejemplo siguiente se explicarán cada uno de ellos. En la imagen que se muestra en B.6, hay un

tweet del usuario Fer Padilla. Vamos a realizar un análisis de las partes de este tweet: el texto en color blanco es el texto en sí del tweet, las palabras escritas en un color azul oscuro hacen referencia a los hashtag que contienen este tweet. Más abajo se puede observar los siguientes iconos:



Figura 6.1: Ejemplo de tweet

- El bocadillo hace referencia al número de comentarios que tiene este tweet, en este caso 2.
- Las flechas girando sobre sí mismas hacen alusión al número de retweet de un tweet (un retweet es publicar el mismo tweet pero escrito por otro usuario), dicho de otra forma etiquetas el tweet en la lista de publicaciones propia.
- El icono del corazón hace referencia al número de favoritos que se ha llevado dicho tweet, es el mismo símil que facebook con los likes.
- El icono de la caja y la flecha es para copiar el enlace del tweet o guardarlo en una lista.

Esta pequeña parte es para que el usuario que lea esto se ponga un poco en contexto de los términos importantes que se utilizan en esta red social.

Capítulo 7

Análisis de herramientas

En este capítulo se va a presentar las herramientas con las que se ha abordado el proyecto: entorno de desarrollo, las librerías y el equipo con el que se ha desarrollado.

7.1. Entorno de desarrollo

El entorno de desarrollo que se a utilizado para la investigación y el desarrollo de software ha sido Pycharm Pro[12]. Pycharm esta creado por JetBrains[13] python, es compatible con Python, JavaScript, CoffeeScript, TypeScript, CSS, contiene una consola de Python integrada en el ide, Pycharm tiene las siguientes versiones:

- **Comunidad:**

Es la versión gratuita del proyecto Pycharm esta tiene menos capacidades que la profesional, editor de Python, depurador gráfico, navegación y refactorización del código, compatibilidad con VCS.

- **Profesional:**

Esta es la versión completa del software que está más enfocado para desarrollo web, contiene además herramientas científicas, marcos de trabajo web Python, capacidades para desarrollo remoto. Todo esto por un precio de 89 dolares anuales.

Gracias a la Universidad de Jaén se ha podido tener una licencia de la versión profesional sin coste ninguno.

7.1.1. ¿Por qué se ha elegido Pycharm?

Las razones por las cuales se ha elegido Pycharm son las siguientes:

- **Compatibilidad:**

Todo el estudio esta se ha hecho sobre un sistema operativo MacOS, por lo que se necesita un entorno de desarrollo que fuera compatible, ya que el editor de texto que trae la instalación de Python es muy pobre.

- **Desarrollo Web:**

Al tener que hacer en el proyecto una página web y tener Pycharm un entorno de desarrollo para web ha sido otro factor para su elección.

- **Subscripción profesional**

Se ha podido optar a una subscripción estudiantil para la versión profesional y tener todas la herramientas disponibles para el TFG.

- **Facilidad de uso**

El entorno es fácil de usar y además tiene una gran comunidad por detrás, la cual contesta cualquier duda sobre el uso de dicha herramienta. Tiene integrado el instalador de paquetes pip el cual se ha utilizado constantemente para la prueba de librerías, no se descarta en un futuro seguir utilizándola para otros proyectos en Python.

- **Herramientas integradas**

El entorno brinda muchas facilidades a la hora de utilizarlo para realizar proyectos:

1. **Vistas integradas**

Por defecto contiene vistas para la manipulación de carpetas y ficheros.

2. **Frameworks**

Soporta Frameworks web, en el caso de este TFG se ha utilizado Flask[14], pero también dan las opciones de Django[15] y Web2py[16].

3. **Herramientas programación**

Depurador, terminal de Python, terminal de sistema, Git y refactorización del código.

4. **Configuración de interfaz**

Se puede elegir diversas opciones de interfaz de usuario, elección de temas, tipo de fuentes, configuración de colores dependiendo del lenguaje que se va a utilizar.

- **Estadística de uso**

Los usuarios que usualmente programan en Python utilizan los siguientes IDE:

- PyCharm Profesional Edición 25 %
- PyCharm Community Edición 17 %
- VS Code 14 %
- Vim 8 %
- IntelliJ IDEA 6 %
- Sublime Text 6 %
- Jupyter Notebook 6 %
- Atom 4 %
- Ipython Notebook 3 %
- Notepad++ 2 %
- Otros 9 %

Por todas estas características, se ha elegido Pycharm Profesional edición como el entorno de desarrollo para este TFG.

7.2. Software y librerías

7.2.1. API de Twitter

La API de Twitter[17] en la que se va a fundamentar el trabajo de investigación esta dividida en diferentes herramientas que Twitter proporciona:

- API estándar

Esta es la versión de la Api que se ha utilizado en el TFG entre sus características más importantes se encuentra:

1. **Recuperar Tweets**

Básico para la investigación que se va a realizar, la descarga se realiza en JSON.

2. **Publicar en tweets**

Añadir tweets en la cuenta de Twitter del desarrollador.

3. **Mensajes directos**

Mandar mensajes directos a otras cuentas y obtener los mensajes que reciba la cuenta del usuario.

4. **Información Usuarios**

Se pueden acceder a la información del usuario, número de seguidores, número de seguidos, cumpleaños, fecha en la que creo una cuenta en Twitter, ver si es un usuario verificado (esta característica es importante a la hora de filtrado de tweets).

5. **Trending topic**

Analizar los Trending topic en busca de grupos de tweets hablando de temas en relación con el objeto de estudio.

6. **Precio**

Su precio es gratuito.

■ **API Premium**

Esta versión de la API por un coste de 150 \$ cada mes, dependiendo del número de consultas, en el caso de no llegar a las 100 solicitudes su precio es de 99 \$. Algunas de las características más importantes de esta versión son:

1. **Historial de búsqueda**

Una gran diferencia entre la versión estándar, que es la utilizada en este TFG, y la versión Profesional es el tiempo en el cual podemos realizar peticiones de búsqueda en la red de Twitter. Puesto que la versión estándar tiene un máximo de antigüedad de los tweets de una semana, mientras que en la versión profesional es de 30 días. Esto no exime que la versión estándar no pueda acceder a los tweets de un usuario, cuando la última vez que escribió fue hace 2 años ese tweet si es analizable.

2. **Actividad de la cuenta**

La versión profesional puede ver las menciones que tenga cada usuario con respecto a los tweets, retweets, favoritos, además de incluir un acceso de más cuentas suscritas con un precio de 339 \$ por cada 25 subscripciones. Cada cuenta es un usuario, el cual desde la Api puede hacer que escriba tweets o siga a otros usuarios. Un ejemplo de este uso son las cuentas de publicidad que siguen a usuarios dependiendo del contenido de sus tweets.

3. **Uso de datos** Da acceso a un mánager para ver análisis de la cantidad de datos descargados.

■ **API Enterprise**

La opción Enterprise es un servicio personalizado al usuario que lo contrata, además de dar unas mejoras en el aspecto de captura de tweet en tiempo real. Las características de esta versión no pueden verse sino se contrata.

Para este TFG, por motivos de precio se ha elegido la versión estándar, ya que es un trabajo teórico-experimental.

7.2.2. **Tweepy**

Tweepy[18] es una wrapper de Twitter, es open source, escrita en Python para tener acceso a la Api de twitter.

Esta librería tiene acceso a los dos módulos más importantes de la Api de Twitter:

- **Twitter Api**

En este módulo, se realizan las peticiones http a la Api de Twitter, búsqueda de palabras clave y de usuarios y descarga de trending topic.

- **Streaming Api**

Streaming es para realizar captura de tweet en tiempo real.

¿Por qué utilizar Tweepy?

La Api de Twitter tiene muchas librerías en Python que hacen el mismo acceso que Tweepy. Las principales librerías que acceden a la Api de Twitter son: Tweepy, Python Twitter Tools, Python-twitter[19], Twython, TwitterAPI y TwitterSearch. Tras analizarlas se eligió Tweepy por estas razones:

1. El número de desarrolladores del proyecto.

- Tweepy 135
- Python Twitter Tools 60
- Python-twitter 109
- Twython 73
- TwitterAPI 15
- TwitterSearch 8

2. En la comunidad de Python es el que tiene más aceptación, por ello en caso de buscar documentación. Un ejemplo sería que habría más ayuda que al utilizar otra librería.

Por estas razones, fue elegido Tweepy como la librería de acceso a la Api de Twitter.

Funcionalidades Tweepy

Principalmente, Tweepy tiene las mismas funcionalidades que la Api de REST de Twitter. Tweepy solo es un wrapper que utiliza REST, entre sus principales funciones tenemos las siguientes:

- Búsquedas por **palabra clave**, se puede realizar una consulta a la red de Twitter, devolviendo un vector con formato JSON, con cada uno de los campos que tiene la estructura de Tweepy.

- Búsquedas de **usuario**: en ellas se obtiene datos referentes al usuario consultado: número de retweet, número de favoritos, tweets escritos y localización.
- **Escribir tweets** en la red: pero solo con la cuenta del desarrollador.
- Búsquedas en **tiempo real**: se puede descargar tweets en tiempo real aunque esto depende del tipo de versión del desarrollador que se este utilizando, pueden ser por palabra clave o por idioma.
- **Operaciones** sobre usuarios: seguir a otros usuarios, retweetear sus propios Tweets, darle favorito, bloquearlos, mandar mensajes privados a cuentas.

Estos son unas de la principales funcionalidades de Tweepy, hay que resaltar que estas son las funciones más básicas, la potencia de estas funciones va a depender del tipo de versión de la Api.

Tipo de datos

- Status, este tipo de estructura se utiliza en respuesta a peticiones de búsqueda de tweets sobre unas palabras clave o consulta en tiempo real.

Estructura:

- **Created at**
Hace referencia a la fecha de creación del tweet, es de tipo String.
- **Id**
Este es el id del tweet, cada tweet tiene uno identificador único, es de tipo Int.
- **Idstr**
Este es el id del tweet, es igual que el anterior con la única diferencia que es de tipo String.
- **Retweeted**
Este parámetro nos indica si el tweet ha sido retweeteado por algún usuario, puede ser True o False.
- Este parámetro nos indica si el tweet ha sido retweeteado por algún usuario, puede ser True o False.
- **Entities**
Este parámetro nos indica si hay hashtag en el tweet, Url, archivos media incrustados en el tweet y de estos últimos todos sus características: url, url del usuario y los diferentes tamaños del archivo media.

- **Número de retweet**

Indica el número de retweet del tweet.

- **Número de favoritos**

Indica el número de favoritos del tweet.

- **Usuario**

Aquí todos los parámetros relacionados con los datos del usuario.

Esto es un resumen de las partes más importantes de la tipo de dato Status. Este dato es un JSON, dicho dato tiene muchísimos valores, de estos muchos son relevantes y otros campos que no tiene sentido realizar un estudio sobre ellos.

- Usuario: Este tipo de datos es devuelto por funciones que mandan peticiones sobre datos de un usuario.

Estructura:

- **Idstr**

Este es el id del usuario, valor de tipo String.

- **Descripción**

Devuelve una cadena de texto con la descripción que el usuario publica en twitter.

- **Name**

Contiene el valor string del nombre del usuario.

- **Protected**

Indica si el usuario esta protegido o no.

- **Created_at**

Muestra la fecha en la que fue creada la cuenta del usuario.

- **Location**

Indica la localización del registro de la cuenta de Twitter.

- **Timezone**

Detalla la zona horaria del usuario.

- **Verified**

Indica si el usuario esta verificado, es un usuario publico.

- **Followerscount**

Contiene el número de seguidores que tiene el usuario.

- **Favouritescount**

Muestra el número de favorito que tiene el usuario.

- **Screenname**

Nombre de la cuenta del usuario.

El tipo de dato usuario contiene 45 parámetros, estos son los más importantes. Dichos parámetros no nombrados están relacionados con la personalización del aspecto de la cuenta de twitter, imagen de avatar, color del texto.

Valores escogidos para análisis

Dado que la Api de Twitter tiene una cantidad muy grande de parámetros, de los cuales muchos no son útiles para nuestro estudio, aquí presento los valores que he escogido para realizar el análisis.

- **Createdat**: La fecha en la que se creo el tweet.
- **Full_text**: El propio texto del tweet.
- **Id_tweet**
El número único que identifica a cada tweet.
- **Id_user**
El número único que tiene cada usuario asignado al crearse.
- **Reetweet_count**
El número de retweet que ha tenido el tweet.
- **Favorite_count**
La cantidad de favoritos que ha tenido el tweet.
- **Verified**
El parámetro que indica si el usuario es verificado.
- **Location**
La localización del tweet.
- **Screen_name**
El nombre de la cuenta del usuario.
- **URLpagina**
El sitio web del usuario.
- **URL_tweet**
Concatenando el nombre del usuario y el id del tweet tenemos la dirección del tweet que estamos analizando.

Con estos datos se realizarán los diferentes análisis después de la captura de tweets.

Primeros pasos

Bueno antes de empezar con Tweepy, se creará un correo solo para el TFG.

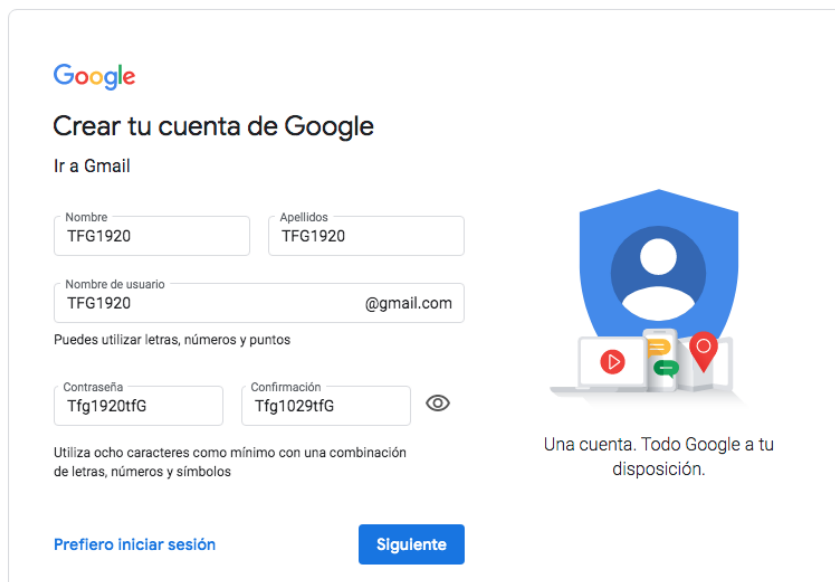
The image shows the Google account creation interface. At the top is the Google logo, followed by the heading 'Crear tu cuenta de Google' and the subtext 'Ir a Gmail'. The form contains several input fields: 'Nombre' (First Name) with the value 'TFG1920', 'Apellidos' (Last Name) with the value 'TFG1920', 'Nombre de usuario' (Username) with the value 'TFG1920' and a placeholder '@gmail.com', 'Contraseña' (Password) with the value 'Tfg1920tfG', and 'Confirmación' (Confirmation) with the value 'Tfg1029tfG'. There is an eye icon for password visibility. Below the password fields is a note: 'Utiliza ocho caracteres como mínimo con una combinación de letras, números y símbolos'. To the right of the form is a graphic of a blue shield with a white person icon, and below it, a laptop displaying various Google services icons (YouTube, Maps, etc.). At the bottom left is a link 'Prefiero iniciar sesión' and at the bottom right is a blue button labeled 'Siguinte'.

Figura 7.1: Usuario de Gmail para las cuentas de la Api de Twitter

Después de la creación de este correo específico para el trabajo se deberá registrarse en Twitter.

Tras crearse la cuenta de Twitter, debe registrarse como desarrollador, en la página de developers de Twitter. Después del registro aparece un botón que se identifica como Create Application, el problema es que se encuentra bloqueado y avisa con este mensaje.

Para conseguir los permisos de desarrollador Twitter se pondrán en contacto con el usuario para saber las intenciones, todo esto por correo electrónico.

Con dichos correos, se informará de los tipos de datos que se utilizará con esta aplicación.



Figura 7.2: Usuario de Twitter

You cannot create apps because your
developer account is pending.

Figura 7.3: Bloqueo por cuenta de desarrollador

La respuesta de los agentes de Twitter es sobre la ciencia de datos para el análisis. En este TFG se usará dicha ciencia de datos y con otro correo preguntan si va a ser de uso comercial. Después se consiguió el permiso para crear aplicaciones en Twitter developer.

Al iniciar la aplicación se deberá completar algunos datos: nombre de la aplicación, descripción de la aplicación, sitio web del usuario. Tras esto, preguntaran sobre nuestro objetivo con los datos de twitter, en este caso solo es descargar tweets.

Con todo esto, ya se ha adquirido la aplicación creada en Twitter developers.

How will you use the Twitter API or Twitter data? All fields are required unless marked optional

In your words

In English, please describe how you plan to use Twitter data and/or APIs. For students and teachers, please include the name of the school, the name of the instructor and the course number (if available). The more detailed the response, the easier it is to review and approve.

My app is for university use and is part of a university degree final project,
In this application we will conduct a study of social events in the application in addition to an analysis of the most relevant tweets and users.


Response must be at least 200 characters ✓

Figura 7.4: Explicación del uso del TFG

Apps > **Twitter-Event**

App details **Keys and tokens** **Permissions**

App details Edit

 **App icon**
Click edit to upload a new icon.

App Name
Twitter-Event

Description
aplicación para el estudio de un evento o hashtag de twitter

Website URL
<https://tevent20231565.wordpress.com>

Sign in with Twitter
Disabled

Callback URL
None

Terms of service URL
None

Privacy policy URL

Figura 7.5: Aplicación creada en la cuenta de developer de twitter

Ahora el siguiente paso es adquirir las claves de acceso a la red de Twitter, para ello haremos clic en la pestaña key and tokens para generarlos.

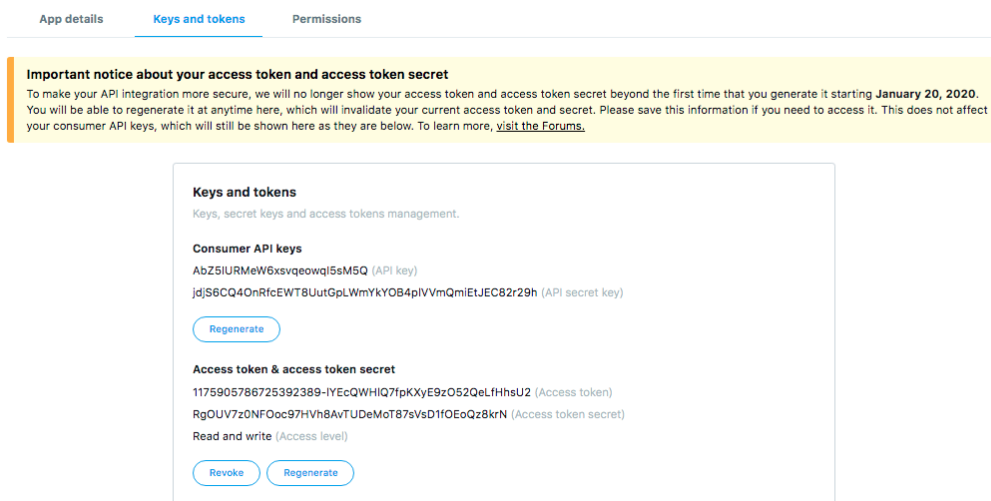


Figura 7.6: Claves de acceso a la Api de Twitter

Estas claves son las que se utilizan para realizar consultas a la red desde la aplicación en Python.

Comienzo con Python

Al iniciar el programa en Python lo primero que se debe hacer es instalar la librería de Tweepy.

Para ello en la consola por medio del comando pip introducimos:

```
pip3 install tweepy
```

Ahora se importará la librería en el código, seguido de esto hay que copiar las claves de acceso en variables y declarar una variable que encapsule las claves.

Las peticiones que se harán a la red de Twitter serán por esta variable api.

```
import tweepy

CONSUMER_KEY = 'AbZ5IURMeW6xsvqeowqI5sM5Q'
CONSUMER_SECRET = 'jdjS6CQ40nRfcEWT8UutGpLWmYkY0B4pLVVmQmiEtJEC82r29h'
ACCESS_TOKEN = '1175905786725392389-lYEcQWHlQ7fpKXyE9z052QeLfHhsU2'
ACCESS_TOKEN_SECRET = 'Rg0UV7z0NF0oc97HVh8AvTUDeMoT87sVsD1f0EoQz8krN'

auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)

api = tweepy.API(auth)
```

Figura 7.7: Claves de acceso a la Api de Twitter

7.3. Equipo hardware

El equipo con el que se ha realizado el TFG es un Macbook pro de 13 pulgadas de finales de 2011 con las siguientes características:

- Procesador: Intel Core i7 2,8 GHz
- Memoria: 8 Gb 1333 MHz DDr3
- Gráfica: Intel HD Graphics 3000 512 MB
- Disco Duro: SSD Samsung EVO 810
- Sistema Operativo: MacOS High Sierra versión 10.13.6



Macbook pro 13" de finales de 2011

Capítulo 8

Captura de datos

La captura de datos se ha distribuido en los siguientes puntos:

- Tipos de consultas
- Funciones de la Api
- Tipos de resultados
- Tipo de modo de tweets.
- Otros argumentos de captura.
- Variables de entorno.
- Funciones de captura.
- Estructura de captura.
- Observaciones de los datos capturados.
- Datos capturados.

8.0.1. Tipos de consultas

En la Api de Twitter se puede encontrar dos tipos de consultas a la red de Twitter. La primera es sobre datos ya escritos en la red, lo que sería una consulta con una duración máxima de una semana de antigüedad. La única forma de escapar a esta restricción es porque se haga una consulta sobre un usuario y este no haya escrito en su tablón desde hace mucho tiempo. El otro tipo de consultas es en tiempo real, aunque en la documentación este escrito que es en tiempo real por los experimentos que se han hecho a lo largo del TFG, los tweets tienen una antigüedad de horas, el menos antiguo que se ha conseguido ha sido de diez minutos.

8.0.2. Funciones de la Api

Hay diferentes tipo de llamadas a la Api depende de la consulta de datos que se requiere recibir. Van desde consultas de estados de usuarios, actualización de usuarios, bloqueo de usuarios, creación de listas. Todos estos métodos que son aplicados sobre cuentas de usuarios o sobre la cuenta de usuario con la que se conecta a la Api han sido obviados, porque no tienen relevancia para el estudio, dicho estudio se ha centrado más en la captura de datos y en su procesamiento.

Estas son las funciones de la Api que se ha utilizado para crear la aplicación:

- `api.search`: devuelve un conjunto de tweet aleatorios, valores de entrada, la palabra clave por la que va a buscar. Este método tiene dos tipos de adquisición de tweets dependiendo la versión antigua o la versión moderna con Cursor:
 - Forma antigua es guardando la búsqueda en una variable y luego recorriéndola.
 - La forma de la versión moderna es por medio por la estructura Cursor.
- `api3.trends_place`: retorna una lista de trending topic de una localización, estas localizaciones tienen un código WOEID. Es el código por el que se busca en la Api, en España solo hay diez ciudades con tales códigos por lo que la búsqueda de lugares exactos no se puede realizar, solo en las ciudades que tengan dichos códigos. Las localizaciones españolas con WOEID[20]:
 - Barcelona
 - Bilbao
 - Las Palmas
 - Madrid
 - Málaga
 - Murcia
 - Palma
 - Sevilla
 - Valencia
- `api.search_users`: devuelve los usuarios que están relacionados con un tema de entrada, son los que más relevancia tienen en Twitter.
- `api.user_timeline`: devuelve los últimos tweets escritos o retweteados de un usuario.

8.0.3. Tipo de resultados

Cuando el programa realiza peticiones a la Api de Twitter existen tres modalidades:

- Recent: esta opción da los tweets más recientes.
- Popular: muestra los tweets con más retweets y favoritos.
- Mixed: hace una mezcla entre las dos opciones.

Teniendo en cuenta el límite de descarga de tweet que tiene la Api para la versión que se ha utilizado, hay que hacer varias peticiones de búsqueda con diferentes tipos de resultados, esto se hace para intentar encontrar el mayor número de tweets distintos.

Búsqueda tiene la opción de mixed, la función Busqueda2 tiene la opción de recent y Busqueda3 con la opción popular.

8.0.4. Tipo de modo de tweets

En las primeras versiones del TFG no se tenía en cuenta el modo de tweets. Al principio funcionaba correctamente, pero en algunos tweets no aparecía todo el texto. Solo aparecía tres puntos seguidos y no podía acceder al texto restante. Hasta que tras mirar la documentación introduciendo extended devuelve todo el tweet completo.

```
Busqueda = api.search(valor,tweet_mode='extended')
```

Así al acceder al campo del tweet que tiene el texto se devolverá el campo completo de este.

8.0.5. Argumentos de captura

Estos son unos argumentos en la captura de tweet, es importante de resaltar para la hora de capturar tweets.

- Lenguaje: para introducir el idioma de los tweets.
- Since y until: podemos introducir intervalos de fecha para realizar las consultas siempre que no superen la semana de antigüedad.
- Count: la cantidad de tweet que se descargan en cada consulta.
- -Filter:retweets: esta opción pide a la api tweets que no sean retweets, así eliminamos todos los tweets repetidos, para no capturarlos.

8.0.6. Variables de entorno

- Ciudades: lista con las localizaciones que tienen código WOEID.
- Códigos: lista con los códigos de cada ciudad.
- Hora: es una variable con la hora del sistema, es el valor primero para la búsqueda del foco del tema de búsqueda.
- Foco: esta contiene el tweet más antiguo que se ha encontrado sobre el tema de búsqueda.
- Conjunto: es el conjunto con los id de tweet que vamos encontrando, sirve de discriminador de tweets repetidos.
- Tweets_usuarios: son los tweets encontrados de los usuarios

8.0.7. Funciones de captura

1. CapturarUsuarios(tema): Complejidad:O(n)

Esta función hace una consulta a la Api de Twitter y le pide los usuarios más relevantes de ese tema. **valores de entrada** es un string con el tema de búsqueda. La razón de esta función es realizar un seguimiento de los usuarios que se ha encontrado y cotejar sus últimos tweet, con el tema de búsqueda general que se ha introducido por el usuario en la interfaz. **Valores de salida** es una lista con los usuarios encontrados.

```
def CapturarUsuarios(tema):  
  
    try:  
        users = api3.search_users(q=tema, lang='es')  
        global usuarios  
        usuarios=[]  
        for user in users:  
            usuarios.append(user.screen_name)  
        print(usuarios)  
        guardarUsuarios(usuarios)  
        return usuarios  
    except tweepy.TweepError:  
        print("Excepcion de limite de tweets descargados")  
        logapi += 1  
        time.sleep(60)
```

2. AnalisisUsuario(usuarios, tema):

Complejidad: $O(n)$ Esta función recorre la lista de usuarios y llama a la función ComprobarTweet, se le pasa como argumentos el iterador y el tema de búsqueda que introdujo el usuario. **valores entrada** son la lista de usuarios encontrada por CapturarUsuarios y el tema inicial de búsqueda del usuario. **valores de salida** ninguno.

```
def AnalisisUsuario(valor,original):  
    if len(valor)>0:  
        for usu in valor:  
            ComprobarTweet(usu,original)
```

3. ComprobarTweet(usuario,original): Complejidad: $O(n)$ realiza una petición a la Api sobre el usuario y le pide el último tweet que se ha escrito.

Al ser esta función periódica en el tiempo va a hacer la consulta muchas veces sobre el mismo usuario, para evitar que se repitan los tweets que devuelva se ha utilizado un conjunto con los id de cada tweet, el id es campo único de cada tweet así se evita tener información repetida.

4. LimpiarTweet: Esta función se encarga de limpiar cada tweet que se ha descargado y que no esta repetido, antes de introducirlo en la lista de tweet realizamos las siguientes operaciones sobre el tweet.

- Se cambia el texto a minúscula.
- Se realiza una búsqueda en el texto, por la función find de Python, se busca si existe la subcadena con el tema de búsqueda del usuario en el tweet. Si la función nos da un resultado distinto a -1, es que la contiene, en ese caso se guarda el tweet.
- Se elimina los enlaces que puedan haber en texto.
- Se elimina los tabuladores del texto ya que este será el carácter separador de nuestro csv con los datos.
- Se quita los retornos de carro.
- Las comillas dobles ya que hay librerías para el análisis de datos, que no toleran en el texto comillas dobles.
- Después quitamos los emoticonos del texto, usuario y la localización del usuario. Con la función deEmojify. Esto es debido a que los usuarios tienen un afán por poner emoticonos en todos los sitios que pueden, tanto texto por sensaciones que quieran imprimir, en el nombre de usuario y por último en la localización por los muchos usuarios que ponen una bandera con su país de procedencia.

- Por último vamos concatenando todas las partes del tweet que interesan, entre ellas tabuladores.
- Por último se guarda en una lista llamada listaUsuarios. Esta es la lista que se mostrarán en el análisis final con los tweets los usuarios buscados por tema.

```
def deEmojify(inputString):
    return inputString.encode('ascii', 'ignore').decode('ascii')

def limpiarTweet(tweet, bandera):
    global tweets, contador
    tw = ""
    tw = str(tweet.created_at)
    bandera = re.sub(r"(?:\\|https?:\\/|\\S+", "", bandera) # quitar
    bandera = re.sub(r"http\\S+\\n\\r", "", bandera) # quitar
    #print(bandera)

    bandera = re.sub(r"http\\S+", "", bandera)

    s = re.sub(
        r"([^\u0300-\u036f]|n(?:!\u0303(?:[\u0300-\u036f])))[\u0300-\u036f]+",
        normalize("NFD", bandera), 0, re.I
    )
    s = normalize('NFC', s)
    s = deEmojify(s)
    s = s.replace('\t', '')
    s = s.replace('\r', '')
    s = s.replace('"', '')
    tw = tw + '\t' + s.replace("\n", "") + '\t'
    tw = tw + '#' + tweet.id_str + '\t'
    tw = tw + str(tweet.retweeted) + '\t'
    tw = tw + '#' + deEmojify(tweet.user.id_str) + '\t'
    tw = tw + str(tweet.retweet_count) + '\t'
    tw = tw + str(tweet.favorite_count) + '\t'
    tw = tw + str(tweet.user.verified) + '\t'
    tw = tw + deEmojify(str(tweet.user.location)) + '\t'
    tw = tw + deEmojify(str(tweet.user.screen_name)) + '\t'
    tw = tw + str(tweet.user.url) + '\t'
```

```

url = 'https://www.twitter.com/' + deEmojify(str(tweet.user.screen_name))
tw = tw + str(url)

user = deEmojify(tweet.user.screen_name)
tweets = tweets + "tweet de: " + user + "\n" + s+ "\n"+"-----"
contador += 1

return tw

```

Para aumentar el feedback con el usuario en esta función se recogen el texto de los tweets y el número que se han descargado. Así mientras que el usuario espera puede ver los tweets que se están descargando y cuantos lleva la ejecución. El motivo de esta función es que realiza las capturas para mostrarlas al usuario y todas las funciones que capturan tweet confluyen en esta función.

```

def ComprobarTweet(usuario, original):
    originalMinuscula= original.lower()
    try:
        ultimo_tweet = api3.user_timeline(screen_name=usuario
                                           , count=1,tweet_mode="extended")

    try:
        status = ultimo_tweet[0]
        minuscula = status.full_text.lower()
        solucion = minuscula.find(originalMinuscula)
        if solucion != -1:
            contador2 = len(susuarios)
            usuarios.add(status.id_str)
            if contador2 < len(susuarios):
                bandera = str(get_tweet_text(status, 1)).replace('\n', '')
                tw = limpiarTweet(status, bandera)
                tweets_usuarios2.append(tw)
    except IndexError:
        print("")
    except tweepy.TweepError:
        print("Excepcion de limite de tweets descargados")
        logapi+=1
        time.sleep(60)

```

5. CapturaTiempoReal(tema) Complejidad: $O(n)$ esta función realiza una consulta a la Api, a diferencia de las que vamos a ver después, esta busca en el tiempo más reciente.

Para utilizar esta función hay que implementar una clase streamListener, es una subclase dentro de Tweepy que realiza peticiones en tiempo real a la Api. El ejemplo de la documentación devuelve tweet en JSON, pero este formato es el menos cómodo a la hora de manipular los datos.

En los cambios que se ha realizado a la clase son un tiempo máximo de búsqueda por las que puede estar en escucha sesenta segundos, cuando pase ese tiempo sale de la ejecución. **Valores de entrada** el tema de búsqueda que ha introducido el usuario y **valores de salida** ninguno.

El texto que esta función nos devuelve puede ser de tweet en modo full_text o estándar los cuales están cortados a un número de tweet, al llegar a este número de caracteres tienen ..., para evitar esto hay que tratar a cada tweet dependiendo del tipo de tweet que es, se implemento la función get_tweet_text que devuelve el texto de cada tweet dependiendo del tipo que es.

```
def get_tweet_text(tweet,number):
    if number == 0:
        try:
            return tweet.extended_tweet['full_text']
        except AttributeError as e:
            return tweet.text
    else:
        return tweet.full_text

def CapturaTiempoReal(original):
    myStream = tweepy.Stream(auth=api.auth
                             , listener=MyStreamListener(time_limit=60))
    myStream.filter(track=[original])
    resultado = myStream.listener.get_contador()
    return resultado

class MyStreamListener(tweepy.StreamListener):
    contadortweets = 0
    def get_contador(self):
        return self.contadortweets
```

```

def __init__(self, time_limit=60):
    self.start_time = time.time()
    self.limit = time_limit
    super(MyStreamListener, self).__init__()
def on_status(self, status):
    if (time.time() - self.start_time) < self.limit:
        contador = len(conjunto)
        conjunto.add(status.id_str)
        bandera = ""
        bandera = str(get_tweet_text(status,0)).replace('\n', '')
        if bandera.find('RT') == -1:
            if contador < len(conjunto):
                tw = limpiarTweet(status,bandera)
                list.append(tw)
                self.contadortweets += 1
        return True
    else:
        print("Se acabo")
        return False

```

6. Búsqueda(tema): Complejidad: $O(n)$ Su propósito es la descarga de tweets pero solo los tweets que se han escrito a lo largo del día actual, así un usuario cuando introduce varias veces el mismo tema de búsqueda se realizaría búsquedas en el mismo día que se introdujeron los datos. **valores de entrada** es el tema de búsqueda general que introdujo el usuario, es para una búsqueda generalizada. **valores de salida** ninguno.

En esta función hay que introducir un nuevo termino: Cursor, es un método para descargar un gran número de tweets, argumentos añadidos del cursor son la fecha de descarga en los parámetros until y since ya antes nombrados, filtrar retweets, tipo de resultado reciente y la cantidad de tweets que contiene el cursor que son 1000.

Hay que recalcar que aunque se ponga de cantidad 1000, no significa que todos esos tweets nos sirvan. En los experimentos que se ha realizado se ha podido observar que la primeras iteraciones al estar el conjunto de id de tweets vacío, guarda 1000 tweets pero cuando se vuelve a lanzar la función esta guarda un número mucho menor de tweets, alrededor de 120 y así la tercera vez baja a menos de 20, esto es porque son tweets del mismo día en el que lanzó la aplicación, además la Api devuelve tweets aleatorios.

```
def Busqueda(valor):
    numero = 0
    global cantidad
    now = datetime.today()
    tomorrow = datetime.today() + timedelta(days=1)
    yearhoy = now.strftime("%Y")
    monthhoy = now.strftime("%m")
    dayhoy = now.strftime("%d")
    yearmañana = tomorrow.strftime("%Y")
    monthmañana = tomorrow.strftime("%m")
    dayhoymañana = tomorrow.strftime("%d")
    hoy = yearhoy + "-" + monthhoy + "-" + dayhoy
    mañana = yearmañana + "-" + monthmañana + "-" + dayhoymañana
    x = hora
    try:
        for tweet in tweepy.Cursor(api.search, q=(valor)
            + ' -filter:retweets', result_type='recent'
            ,tweet_mode='extended'
            ,lang='es',since=hoy, until=mañana).items(1000):
            contador = len(conjunto)
            conjunto.add(tweet.id_str)
            if contador < len(conjunto):
                #limpiamos el texto
                bandera = str(get_tweet_text(tweet,1)).replace('\n', '')
                tw = limpiarTweet(tweet,bandera)
                if tweet.created_at < x:
                    x = tweet.created_at
                    global foco
                    foco = tw
                    list.append(tw)
                    numero += 1
    except tweepy.TweepError:
        print("Excepcion de limite de tweets descargados")
        logapi2 +=1
        time.sleep(60)

    return numero
```

7. Busqueda2(tema): Complejidad: $O(n)$ Esta función es muy parecida a la anterior solo cambia algunos detalles, los resultados son del tipo mixta. Esa opción busca tweets tanto populares como recientes, además esta función no tiene limitaciones de tiempo busca a lo largo de una semana de antigüedad de los tweets. **valores de entrada** el tema general de búsqueda, **valores de salida** un contador con los tweets nuevos encontrados.

```
def Busqueda2(valor):
    numero = 0
    x = hora
    try:
        for tweet in tweepy.Cursor(api2.search, q=(valor)
            + ' -filter:retweets', result_type='mixed'
            ,tweet_mode='extended',lang='es').items(1000):
            contador = len(conjunto)
            conjunto.add(tweet.id_str)
            if contador < len(conjunto):
                bandera = str(get_tweet_text(tweet,1)).replace('\n', '')
                tw = limpiarTweet(tweet, bandera)
                if tweet.created_at < x:
                    x = tweet.created_at
                    global foco
                    foco = tw
                    list.append(tw)
                    numero += 1
    except tweepy.TweepError:
        print("Excepcion de limite de tweets descargados")
        time.sleep(60)
    return numero
```

8. Busqueda3(tema): Se implementó una versión de la busqueda2 pero con el tipo de dato popular, pero en todas las veces que se ha realizado la búsqueda nunca ha devuelto ningún valor, por eso se ha descartado su uso en la aplicación debido al consumo de la Api. El límite de descarga es un valor constante que hay que tener en cuenta, en el caso de pasarse del límite de la Api, entra en un estado bloqueo durante un tiempo.


```
def Busqueda3(valor):
    numero = 0
    global cantidad
    x = hora
    try:
        for tweet in tweepy.Cursor(api2.search, q=(valor)+
            ' -filter:retweets', result_type='popular'
            ,tweet_mode='extended',lang='es').items(1000):

            contador = len(conjunto)
            conjunto.add(tweet.id_str)

            if contador < len(conjunto):

                bandera = str(get_tweet_text(tweet,1)).replace('\n', '')
                tw = limpiarTweet(tweet, bandera)

                if tweet.created_at < x:
                    x = tweet.created_at
                    global foco
                    foco = tw

                list.append(tw)
                numero += 1
    except tweepy.TweepError:
        print("Excepcion de limite de tweets descargados")
        time.sleep(60)
    return numero
```

9. `AlertaPorHashtag(valor)`: Complejidad: $O(n^2)$ Es la función encargada de realizar un seguimiento de los hashtag que hay en las localizaciones españolas con WOEID y mandar una búsqueda con el hashtag que contiene el valor.

```
def AlertaPorHashtag(valor):
    global codigos
    for c in range(10):

        try:
            trends1 = api3.trends_place(id=codigos[c])
            data = trends1[0]
            trends = data['trends']
            names = [trend['name'] for trend in trends]
            for val in names:
                originalMinuscula = valor.lower()
                minuscula = val.lower()
                busqueda = minuscula.find(originalMinuscula)
                if busqueda != -1:
                    return minuscula
            return ""
        except tweepy.TweepError:
            print("Excepcion de limite de tweets descargados")
            time.sleep(60)
```

Ejemplo: El usuario introduce de búsqueda virus y en Sevilla hay un hashtag llamado Coronavirus al contener la palabra virus, la función llama Búsqueda y le pasa de parámetro Coronavirus, para todas estas funciones se ha tenido que tener en cuenta el límite descarga de datos de la Api, colocando una excepción por si la función sobrepasara seguida de una parada de la ejecución del programa de un minuto, esto es para darle tiempo a las tres claves a recuperarse.

8.0.8. Estructura de captura

En este apartado se va a presentar la estructura de llamadas a funciones y cada cuanto tiempo se van repitiendo cada una de ellas. La intención de esta estructura es para que el límite de cada clave se vayan reponiendo durante el tiempo de ejecución y no se lancen funciones como `AlertaPorHashtag` cada minuto, porque la variable que más hay que controlar es la

caché de cada clave. Además de que los hashtag tardan en cambiar unos minutos, de estar en trending topic a desaparecer. Este método también es para intentar realizar el mayor número de búsquedas a lo largo del tiempo de captura.

```
CapturaTiempoReal(tema)          estas cuatro llamadas son
Busqueda(tema)                   las primeras para no hacer
Busqueda2(tema)                  esperar al usuario
AnalisisUsuario(usuario, tema)
AlertaPorHashtag(tema)
SI valor != "":
    print("valor encontrado en Hashtag", valor)
    Busqueda(valor)
```

```
Mientras (tiempo actual - tiempo ejecutandose) < limite y abortar != 1:
```

```
SI tiempoejecutandose igual 3 * 60 y abortar != 1:
```

```
    CapturaTiempoReal(tema)
    Busqueda(tema)
```

```
SI tiempoejecutandose > 5 * 60 y abortar != 1:
```

```
    Busqueda2(tema)
```

```
SI b - d > 9 * 60 y abortar != 1:
```

```
    SI len(usuario) != 0 o usuario != None:
        AnalisisUsuario(usuario, tema)
        valor = AlertaPorHashtag(tema)
    SI valor != "":
        print("valor encontrado en Hashtag", valor)
        Busqueda(valor)
```

Las cuatro primeras llamadas son porque la caché de la Api está completa, justo después cuando entra en el bucle estará funcionando hasta que llegue al límite de tiempo (en segundos), que haya puesto el usuario en la entrada de datos.

Dentro del bucle se ha elegido los minutos en los que se ejecutan cada función por las siguientes razones:

- **CapturatiempoReal:** cada 3 minutos para que el tiempo de la Api se restaure el límite y para que haya usuarios que a corto plazo hayan escrito sobre el tema de búsqueda.
- **Busqueda(tema):** cada 3 minutos se lanza CapturatiempoReal,seguido se ejecuta Busqueda, porque cada una estas funciones utiliza dos claves distintas de Api, esto es porque cada utiliza unas claves de Api distintas.
- **Busqueda2:** se ejecuta cada 5 minutos, para en el caso de superar el límite de descarga por la Api, el tiempo de bloqueo es de 1 minuto, así al terminar este tiempo no perjudica la ejecución de las otras funciones.
- **AnalisisUsuario y AlertaPorHashtag:** se ejecutan cada 9 minutos porque los hashtag tardan tiempo en cambiar, lo mismo para la búsqueda de tweets nuevos en los usuarios capturados, la frecuencia en los que usuarios escriben tweets no suele ser menor a 9 minutos.

La variable abortar es un botón que tiene el usuario en la interfaz web por si necesita parar la ejecución de la captura.

Una vez que el bucle acabe, guardará los datos descargados en un csv y después empezará la parte del análisis.

8.0.9. Observaciones de los datos capturados

La cantidad de tweets que se puede descargar de un tema específico depende de varios factores:

Factores temporales: si se está hablando sobre un tema en Twitter y en ese momento se pone a capturar, se conseguirán un gran número de tweets, pero hay que tener en cuenta que el tipo de tema de conversación, un ejemplo:

Una noticia: las noticias tienen menos cantidad de información nueva debido a que por lo general los usuarios de Twitter tienen la tendencia de realizar retweet. Esto hace que el número de tweets referente a un tema se incremente pero con tweets repetidos, haciendo que la captura tenga un contenido bajo.

En estos casos como por ejemplo búsqueda de incendios, la mayoría de los tweets son retweets sobre la noticia. Los usuarios que escriben estos tweets suelen ser páginas web de

noticias como trafficspain02, InfoBomberos, carrosdebombero, este tipo de usuarios escriben la noticia y los seguidores realizan retweet o dan favorito al tweet. Esta tendencia hace que no haya información de interés por parte de los usuarios que no son el principal medio de la noticia.

Una tendencia: Es en estos casos cuando no es una noticia, sino es un gran número de usuarios dando información sobre un tema. En estos días de cuarentena se ha realizado diferentes experimentos sobre el tema del coronavirus. Al ser una crisis mundial los usuarios tienen la tendencia a escribir y dar opinión sobre dicho tema, una captura sobre un tema que es tendencia hace que el número de tweets se dispare a más de 10000 tweets, un ejemplo con este tema, un día que ha sido tendencia en 2 horas se ha capturado casi 40000 tweets distintos tanto de usuarios verificados y usuarios estándar. Otro ejemplo distinto es las revueltas de Barcelona que hubo este año. Ese experimento dio alrededor de 35000 tweets en dos horas. Podemos sacar en claro que el usuario que estudia el análisis de un evento en Twitter, ya sea una catástrofe o otro tema tiene que realizar un escaneo de las tendencias en las principales localizaciones españolas con código WOEID, para en ese momento realizar la captura de datos.

Tendencias periódicas: Hay temas que son recurrentes en el tiempo y que el usuario que va a realizar la captura de datos puede anticiparse, a que sean tendencia en España para realizar la captura.

Como ejemplo partidos de fútbol que son semanales, una conferencia del presidente del gobierno que se anuncia con tiempo, el lanzamiento de una película, en este tipo de tendencia el usuario debe realizar una investigación previa al lanzar la captura. La investigación es necesaria debido que la Api de Twitter sus consultas tienen una antigüedad de una semana.

Factores el interés sobre el tema: Aunque el tema de búsqueda que se vaya a realizar el análisis para el usuario sea importante, aquí dependemos de los usuarios de Twitter, un ejemplo sería programas de televisión, los cuales tienen un tipo de usuarios que los ven. Esto hace que la riqueza de la información que se puede adquirir sea pobre, dado que los usuarios que ven ese programa tienen mismos gustos. Otro ejemplo discos de música que son tendencia, pero a todas las personas no les gusta la misma música, si jBalbin saca un disco de requeeton a ese tipo de personas que les guste ese tipo de música crearán una interacción con ese tema en Twitter, pero las personas que no les guste ese estilo no crearán contenido que analizar.

En el caso de las catástrofes las personas que les gusta el medio ambiente realizarán un seguimiento de estas noticias y en el caso de ocurrir algo publicarán tweets con sus propias opiniones sobre el tema.

Este factor hace que buscar sobre un tema con poco interés por parte de los usuarios hace que haya poca cantidad de captura.

Factores geográficos: debido a que la gente utiliza twitter en diferentes situaciones geográficas esto hace que los temas de estudio sean nacionales, las excepciones son temas de interés mundial.

Un ejemplo es cuando Escocia tuvo su referéndum político, el número de tweets que se creó de ese tema fueron a nivel nacional en Reino Unido, pero un tema como el coronavirus crea más información al ser una catástrofe mundial.

Por eso se hace hincapié en que el usuario debe de realizar un análisis previo, sobre el tema debe buscar donde se va a hablar sobre el tema de captura.

Factores generales en las capturas: En las capturas realizadas cada vez que se lanzan tanto las funciones de búsqueda de tweets pasados como en tiempo real el tamaño de la captura, las primeras veces suele ser muy grande de 1000 tweets de descarga el porcentaje de tweets únicos suele ser del 90 %. Las siguientes veces que se lanza la captura el número de tweets de los 1000 tweets de descarga suele quedarse alrededor del 10 %, por este motivo el estudio previo al realizar el análisis es primordial debido a que Twitter va a devolver con cada consulta un número de tweets pero la Api no sabe si esos tweets ya te han sido dados con anterioridad, para solucionar este tema el uso de conjuntos de datos para prescindir de los repetidos y del uso continuado en el tiempo de las funciones de búsqueda da un gran número de tweets descargados.

Otro factor importante es el tema de captura, debido a que la Api de Twitter busca por palabra clave, cada tweet tiene la palabra por la que se hace la consulta a la Api. Esto hace que el poner un tema sea muy complejo en la captura. Por ejemplo, si se quiere saber lo que opinan los españoles de la gestión del tema del coronavirus que ha tenido Pedro Sánchez.

Si la consulta es: 'opinión de los españoles sobre Pedro Sanchez' los tweets que descargue la Api contendrán esa frase introducida, lo que da de resultado que la cantidad sea muy pobre. Una forma mejor de utilizar las consultas es realizar una búsqueda general, en este caso sería coronavirus luego a los datos que se capturen aplicar una búsqueda de unos subvalores que serían: 'opinión, Pedro Sánchez'.

La elección de un tema general de búsqueda y de unos subvalores idóneos harán más fructífera la captura y posteriormente el análisis de los datos.

8.0.10. Datos capturados

Al acabar la captura de datos se generan dos csv:

- Fichero.csv Este fichero es el grueso de los datos descargado que tiene el siguiente formato:
 - **Fecha:** La fecha de cada tweet.
 - **Full_text:** El texto completo de cada tweet.
 - **Id_tweet:**

El id único de cada tweet.

- **Id_USER:**
El id del usuario es un valor único.
- **RetweetCount:**
Número de retweet que ha tenido este tweet.
- **FavoriteCount:**
Número de favoritos que tiene cada tweet.
- **UserVerified:**
Valor booleano que indica si el usuario es verificado o es un usuario estándar.
- **localización:**
La localización del usuario.
- **NombreUsuario:**
El nombre que tiene el usuario en la red de twitter.
- **URLpagina:**
La url de la página que escribió el tweet.
- **URLTweet:**
El enlace de cada tweet.

Este fichero contiene los datos de las funciones de Busqueda, Busqueda2, AlertaPorHashtag y CapturaTiempoReal.

- **Foco.csv**

Aquí se separa el tweet el más antiguo que se encuentra en la captura de datos, tras separarlos por eficiencia para no tener que comparar cada uno de los tweet, es más eficiente a medida que se va capturando e ir comparando.

- **TweetsUsuarios.csv**

Contiene los tweets de los usuarios relacionados con el tema de búsqueda. Al empezar la captura, se hace una petición a Twitter para obtener los usuarios más relevantes con el tema de búsqueda. En este archivo están los tweets que han escrito estos usuarios que tienen relación con el tema de búsqueda general.

Capítulo 9

Análisis Datos

Después de realizar la captura de los datos sobre los valores introducidos por el usuario, los ficheros que se han creado en el proceso anterior tienen que ser analizados.

9.0.1. Librerías de detección

- Spacy

Spacy[21] es una biblioteca de procesamiento del lenguaje natural de Python, esta biblioteca sirve para realizar un análisis sobre datos reales.

La librería se ha utilizado el corpus `es_core_news_md` este modelo contiene los datos relacionados con organizaciones, localidades y personas públicas. Se carga en una variable `nlp2`

```
nlp2 = spacy.load("es_core_news_md")
valor = nlp2(texto)
```

En la variable `valor` se guarda una lista con cada palabra de texto analizada por Spacy, la cual indica si es una localidad, organización o una persona.

- NLTK

Son un conjunto de herramientas de lenguaje natural, tiene un conjunto de herramientas estadísticas, conjunto de palabras vacías para comparar textos, para sacar lo más importante de estos. Otro de las funcionalidades de NLTK[22] es poder lematizar palabras, quedarnos con la raíz de cada palabra esto ayuda en el caso de realizar búsquedas de términos.

Cada palabra de nuestra lengua puede tener distinta conjugaciones, el introducir como valor de búsqueda cada conjugación de cada palabra es algo inadmisibles debido al gran conjunto de datos que tendríamos, esto haría que el tiempo de computo se elevara. Por ello a la hora de buscar valores en el conjunto de datos, se realiza una búsqueda por la función `find` de Python, la idea es buscar subcadenas idénticas dentro del texto del tweet.

9.0.2. Librerías de agrupamiento

Las librerías de agrupamiento tienen como finalidad crear particiones de un conjunto de datos dependiendo de una distancia o una similitud.

En este proyecto se ha utilizado el algoritmo K-means[23] que esta dentro de la biblioteca `sklearn`, `sklearn` es un proyecto de software libre que se basa el aprendizaje automático.

K-means es un algoritmo que subdivide un conjunto en partes las cuales cada una de estas tiene una cercanía a un valor medio, este algoritmo tiene otros nombres k-medias o algoritmo de Lloyd, el problema de este algoritmo es que necesita el número de cluster que se va a dividir el conjunto completo.

Si se tiene un conjunto de 1000 tweets, se necesita un método por el cuál elegir el número de particiones que se va a utilizar, para esto hay varios métodos:

- Dendrogramas:

Es una representación gráfica en la que se muestra un árbol raíz en que se va ramificando el conjunto de datos.

En el ejemplo se ve que hay tres subconjuntos claros: rojo, verde y turquesa. Si se cortase por una distancia euclídea de 25, se obtendría 3 subconjuntos. si se bajase a una distancia de 5, se obtendría 12 subconjuntos.

- El método Gap:

En este método se busca la mayor distancia entre los subgrupos creados para así conseguir el óptimo número de cluster para aplicar k-means.

- El método del codo (Elbow Method):

El método del codo aplica `kmeans` con varios número de cluster sobre el conjunto de datos, se define como inercia la suma de las distancias al cuadrado de cada objeto del cluster a su centroide:

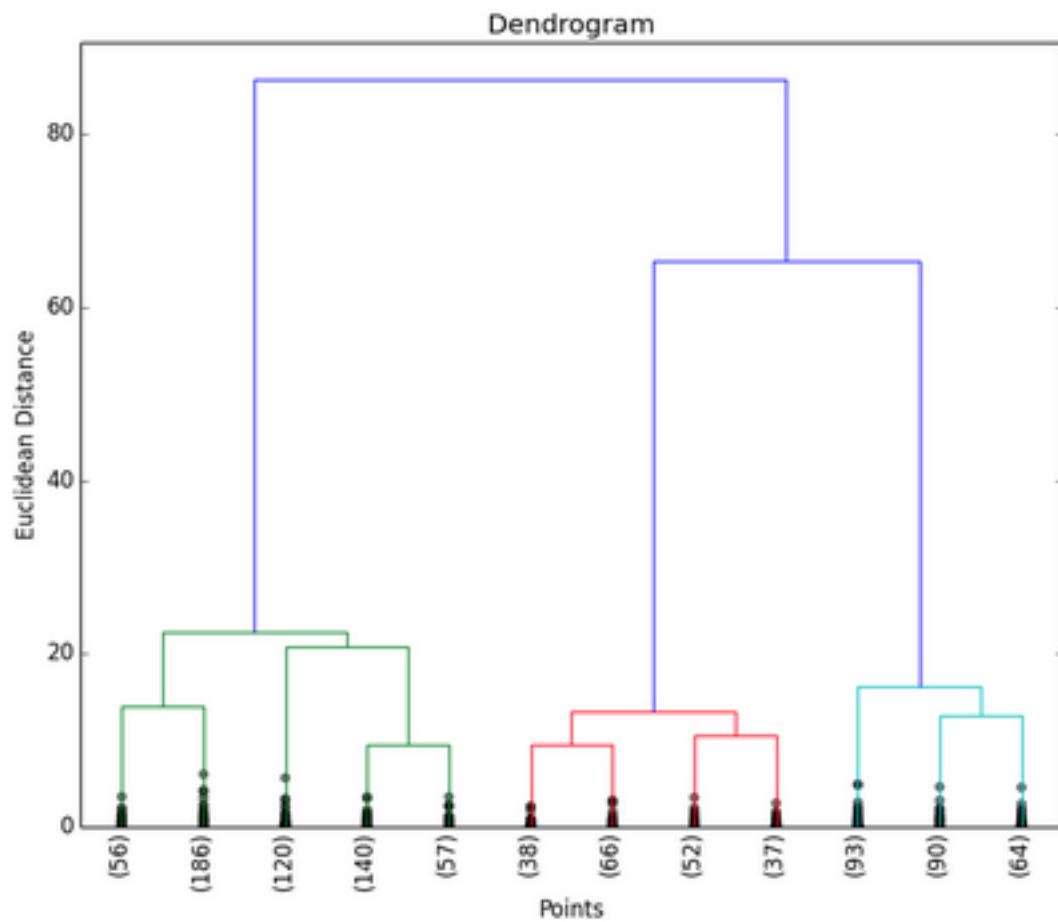


Figura 9.1: Dendrograma de ejemplo

$$Inercia = \sum_{i=0}^N \|x_i - \mu\|^2$$

Figura 9.2: Formula de la inercia, aplicada en Kmeans

Este método busca el cambio más brusco entre las distancias de los cuadrados empleando una gráfica se puede observar mejor estos cambios. Para el desarrollo de la gráfica se ha utilizado matplotlib y para realizar el cálculo la biblioteca kneed que busca en la gráfica el punto con más curvatura y el propio kmeans.

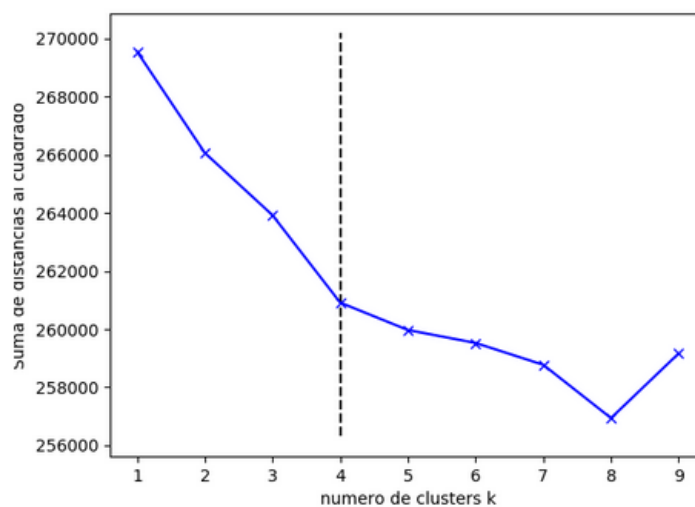


Figura 9.3: Ejemplo de método del codo

En la gráfica de ejemplo se tiene como resultado que el el número óptimo para este conjunto de datos es de 4 cluster.

9.0.3. Palabra relacionadas con el tema

Unos de los problemas que se ha encontrado, ha sido la forma de encontrar palabras relacionadas con el tema principal. Esto es necesario debido a que hay palabras con doble sentido, un ejemplo seria la palabra incendio esta palabra tiene varias connotaciones, una relacionada con el acto de destruir materiales por su combustión y otra puede ser relacionada con un estado de ánimo.

- El teatro fue desalojado por un incendio en el sótano
- Tú no sabes lo que es vivir en un continuo incendio, pensando que no existe nada antes de Dios

Debido a este problema, se ha investigado formas de encontrar palabras relacionadas, la primera opción que se contempló fue la búsqueda de sinónimos de esa palabra. Se encontró la biblioteca nltk que contiene los sinónimos de palabras, el problema fue cuando los idiomas en los que trabaja esta biblioteca son los siguientes:

- als
- arb
- cmn

- dan
- eng
- fas
- fin
- fra
- fre
- heb
- ita
- jpn
- mcr
- msa
- nor
- pol
- por
- tha

En la lista faltaría "spa" para utilizar el idioma español, aun así se realizó una prueba en un pequeño script y como se esperaba dio error con el lenguaje español.

```
Traceback (most recent call last):
  File "/Users/davidbaudetmoreno/PycharmProjects/pruebapartir/main.py", line 23, in <module>
    print ([el.lemma_names('spa') for el in w.synsets('banco')])
  File "/Users/davidbaudetmoreno/PycharmProjects/pruebapartir/venv/lib/python3.7/site-packages/nltk/corpus/util.py", line 123, in __getattr__
    self._load()
  File "/Users/davidbaudetmoreno/PycharmProjects/pruebapartir/venv/lib/python3.7/site-packages/nltk/corpus/util.py", line 88, in __load
    raise e
  File "/Users/davidbaudetmoreno/PycharmProjects/pruebapartir/venv/lib/python3.7/site-packages/nltk/corpus/util.py", line 83, in __load
    root = nltk.data.find("{}{}".format(self.subdir, self._name))
  File "/Users/davidbaudetmoreno/PycharmProjects/pruebapartir/venv/lib/python3.7/site-packages/nltk/data.py", line 701, in find
    raise LookupError(resource_not_found)
LookupError:
=====
Resource wordnet not found.
```

Figura 9.4: El lenguaje español no está en nltk

Otra opción fue realizar webscrapping[24] en páginas que encontraban sinónimos de palabras, pero el problema de que las palabras pueden tener doble sentido, sigue habiendo el problema lo único que se hace es acrecentarlo.

Estos son sinónimos de incendio:

fuego, siniestro, catástrofe, arrebato y pasión. Si se realiza un análisis de las palabras que nos devuelven, las páginas web se observa que fuego si nos puede servir como palabra relacionada con incendio, siniestro puede ser un adjetivo para una situación, persona o otro tipo de accidente, arrebato y pasión tiene más relación con sentimientos personales que con un incendio.

Por esta razón la siguiente opción fue los tesauros, (–pregunte a mi tutor que me guiara un poco sobre este tema, el me paso unas páginas web para realizar peticiones a tesauros–deberías quitarlos).

Los Tesauros de uso general que estudie fueron los siguientes:

- SKOS: Tesoro de la UNESCO[25]. Al realizar una consulta con la palabra 'incendio' da los siguientes resultados:

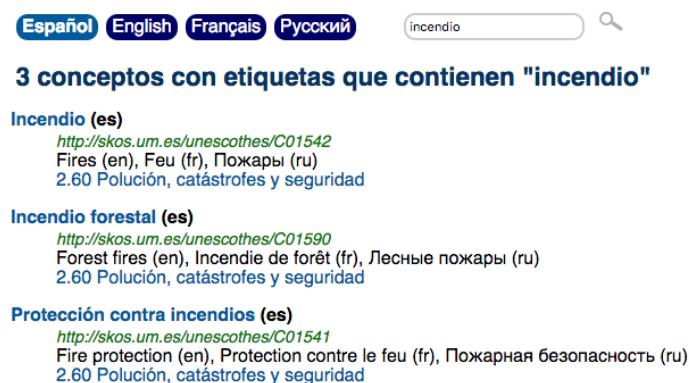


Figura 9.5: Ejemplo del tesoro de la Unesco

Devuelve pocos valores para la palabra incendio y catástrofe puede referirse a inundaciones, terremotos, etc.

- Tesoro SPINES[26]. Los resultados de la consulta tras introducir incendio.

Hay el mismo problema, las palabras que aparecen con el termino incendio están relacionadas pero las nuevas palabras relacionadas como armas incendiarias. Son palabras compuestas en Twitter tendría que salir la palabra armas incendiarias en el caso de

incendios**Términos genéricos**TG ↑ [desastres](#)**Términos específicos**TE4 ↓ [incendios forestales](#)**Términos relacionados**TR ↔ [accidentes de trabajo](#)TR ↔ [armas incendiarias](#)TR ↔ [combustion](#)TR ↔ [explosiones](#)TR ↔ [explosivos](#)TR ↔ [materiales resistentes al fuego](#)TR ↔ [seguro contra incendios](#)

Figura 9.6: Ejemplo del tesauro SPINES

encontrarlo en un tweet. Se busca como solución sustantivos relacionados con la palabra incendio en este caso, luego esta el problema de las consultas realizar peticiones por webscrapping a una web, tienen un límite de pocas peticiones, en el caso de que este proyecto se utilizará como una aplicación final, no sería útil por las limitaciones de las propias páginas web.

- Tesauro de la universidad de Madrid[27]

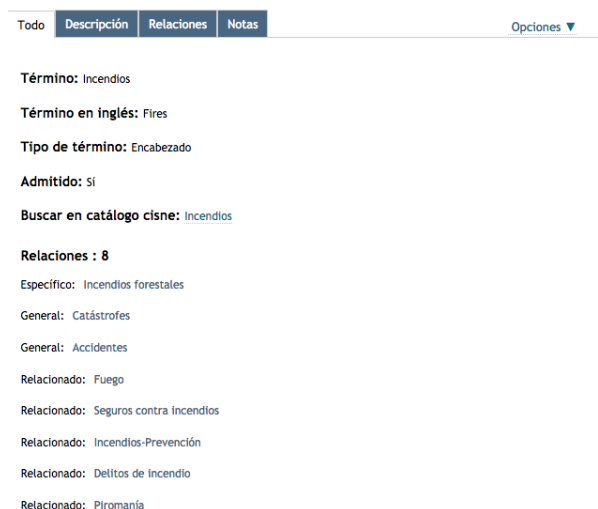


Figura 9.7: Ejemplo del tesauro de la universidad de Madrid

Este tesauro da como resultados palabras que contienen incendio, tienen relación la

búsqueda pero la única que nos serviría para el análisis sería Fuego.

Visto que los tesauros generales que hay online dan resultados que tienen poca relación con la palabra de búsqueda hay que contemplar otras opciones.

Otra opción que se podría tomar sería crear nuestro propio tesoro, los inconvenientes sería el tiempo en crearlo, este fue el motivo para desechar esta idea.

Por último, se buscó enciclopedias en las que se pudiera realizar peticiones de búsqueda dado que en los significados de las palabras hay palabras relacionadas, como solución encontré la Api de Wikipedia[28] para Python.

9.0.4. Api Wikipedia

Al utilizar una Api para realizar las consultas ya no solo se puede buscar valores sobre catástrofes, se puede abstraer un poco más el proyecto a cualquier ámbito.

- Consulta con incendio:

Un incendio es una ocurrencia de fuego no controlada que puede afectar o abrasar algo que no está destinado a quemarse. Puede afectar a estructuras y a seres vivos. La exposición de los seres vivos a un incendio puede producir daños muy graves hasta la muerte, generalmente por inhalación de humo o por desvanecimiento producido por la intoxicación y posteriormente por quemaduras graves.

== Origen de incendio ==

En los edificios, los fuegos (no procede el término incendios) pueden empezar por causas muy variadas: fallos en las instalaciones eléctricas o de combustión (como las calderas), escapes de combustible, accidentes en la cocina, niños jugando con mecheros o fósforos, o accidentes que implican otras fuentes de fuego, como velas y cigarrillos. El fuego puede propagarse rápidamente a otras estructuras, especialmente en aquellas que no cumplen las normas básicas de seguridad haciendo que por ello pase a etapa de incendio, ya que el incendio es la siguiente etapa del fuego descontrolado ante la ausencia de los sistemas de protección pasiva y activa de los incendios.

Este sería la respuesta a la consulta integra ahora de aquí nos quedamos solo con los sustantivos y luego los lematizamos.

Solo los sustantivos:

Exposición, combustión, muerte, intoxicación, daños, niños, cigarrillos, seguridad, ausencia, origen, seres, incendio, causas, incendios, desvanecimiento, ocurrencia, etapa,

sistemas, calderas, accidentes, mecheros, cocina, escapes, término, edificios, fuegos, humo, instalaciones, inhalación, combustible, quemarse, fósforos, velas, fuego, normas, protección, fallos, fuentes, estructuras.

Otro ejemplo con coronavirus :

Orthocoronavirinae, comúnmente conocidos como coronavirus, es una subfamilia de virus ARN monocatenario positivos perteneciente a la familia Coronaviridae. El tamaño de sus genomas varía entre los 26 a 32 kilonucleótidos, siendo así los más grandes dentro de los virus ARN. Se subdivide en los géneros Alphacoronavirus, Betacoronavirus, Gammacoronavirus y Deltacoronavirus. Estos incluyen genogrupos filogenéticamente similares de virus con una nucleocápside de simetría helicoidal envuelta. Se les llama coronavirus por la corona de puntas que se ve alrededor de la superficie del virus. Miden entre 120 y 160 nm de diámetro. Los coronavirus pueden infectar aves y mamíferos produciendo una serie de enfermedades respiratorias y digestivas, muchas de ellas letales trayendo como consecuencia serios perjuicios en la avicultura y la ganadería; también pueden infectar al ser humano causando enfermedades que van desde el resfriado común hasta enfermedades más graves, como bronquitis, bronquiolitis, neumonía, el síndrome respiratorio de Oriente Medio (MERS-CoV), síndrome respiratorio agudo grave (SARS-CoV), entre otras.

Solo los Sustantivos:

Virus, orthocoronavirinae, síndrome, cov, arn, mers, puntas, nucleocápside, medio, tamaño, diámetro, subfamilia, mamíferos, kilonucleótidos, bronquiolitis, oriente, coronavirus, ganadería, genogrupos, familia, bronquitis, sars, monocatenario, enfermedades, helicoidal, betacoronavirus, coronaviridae, alphacoronavirus, resfriado, aves, consecuencia, grave, avicultura, neumonía, perjuicios, deltacoronavirus. géneros, corona, serie, superficie, gammacoronavirus, agudo, genomas.

La consulta devuelve muchos valores que son candidatos de ser encontrados en los tweets, ahora que al pasar todos los tweets hay que encontrar una manera de resaltar los tweets, los cuales tienen correspondencia con los valores de wikipedia, para esto lo mejor es un sistema de puntuación, dotando a cada tweet de una puntuación que sera más elevada conforme más palabras contenga de Wikipedia, esto luego nos dará la posibilidad de representarlo gráficamente.

9.0.5. Búsqueda de valores en los tweets

La búsqueda de valores introducidos por el usuario tiene como finalidad buscar dentro de los tweets descargados, tweets que contengan palabras en específico, como se ha explicado antes las consultas a la Api de Twitter, se realizan por palabra clave. Si se realiza una consulta

con la palabra "#coronavirus", esta consulta devolvería valores que contengan este hashtag, ¿cual es el problema?, hay una tendencia en Twitter de que usuarios que busquen que se compren sus productos o ganar seguidores, es que introducen publicidad dentro de hashtag, estos tweets dentro de un hashtag no tienen relevancia con el tema que representa el hashtag, por ello si realizamos el mismo trato de la información que en el apartado de Wikipedia, se obtendría una puntuación de cada tweets respecto a los subvalores de búsqueda.

Un ejemplo:

Al realizar una búsqueda con el termino gobierno dado que en la cuarentena están tomando iniciativas contra el virus, ahora interesa saber que se opina sobre Pedro Sánchez o el gobierno.

Si la consulta tuviera como entrada solamente "#Pedro Sánchez" todos los tweets que se tendrían, serían tweets en los que apareciera Pedro Sánchez, entonces las referencias hacia el gobierno en general quedarían descartadas.

Por ello se hace una búsqueda más general con el tema gobierno y se introduce unos valores de subbúsqueda en este caso Pedro, Sánchez, muertes y coronavirus.

En el ejemplo se ha encontrado este tweet con puntuación de dos puntos:



Figura 9.8: Ejemplo de tweet con puntuación de dos puntos

Si se pincha en el enlace, se abre el tweet.

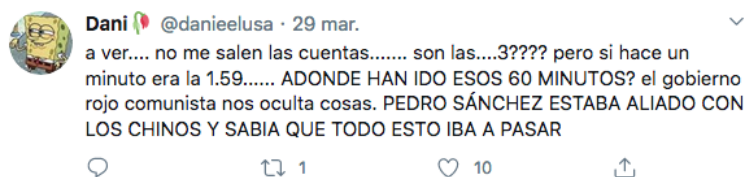


Figura 9.9: Tweet después de pinchar en enlace

Con el método de la gráfica y la puntuación se intenta resaltar los tweets con más relación de la búsqueda del usuario, para mejorar la eficiencia de este método se realiza con conjuntos de palabras, las palabras antes de comparar se lematizan para quedarnos con la raíz de cada palabra así encontramos más resultados y nos quitamos los problemas de las distintas conjugaciones de cada palabra.

9.0.6. Análisis Morfosintáctico

Para el análisis morfosintáctico se estudiaron diferentes técnicas para llevar a cabo el análisis de los tweets que se descarguen de un tema concreto. Al principio el estudio el análisis morfológico de los tweets utilizando nltk el problema es la representación de esta información debido a que los resultados son líneas de texto las cuales su comprensión es difícil da para cada palabra una etiqueta con su valor morfológico, para el estudio sintáctico de los tweets las librerías estudiadas necesitaban de una gramática de entrada para realizar la conversión a árbol.

```
>>> sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
>>> parser = nltk.ChartParser(groucho_grammar)
>>> for tree in parser.parse(sent):
...     print(tree)
```

La salida que se muestra es un nodo padre S del que todos los demás nodos hijos cuelgan de este se van agrupando dependiendo de el sistema que tiene la oración. En el ejemplo el sistema NP contiene an elephant, estos dos elementos hacen un sintagma nominal.

```
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant))))
    (PP (P in) (NP (Det my) (N pajamas))))
  (S
```

```

(NP I)
(VP
  (V shot)
  (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))

```

Además de unas funciones propias de la librería para mostrar el árbol y así facilitar la comprensión del usuario de la estructura de las oraciones, los resultados eran buenos pero la forma con la que se puede manipular la información resultante es muy pobre debido a que las funciones con las que se representan la información de forma gráfica en árbol son propias de nltk.

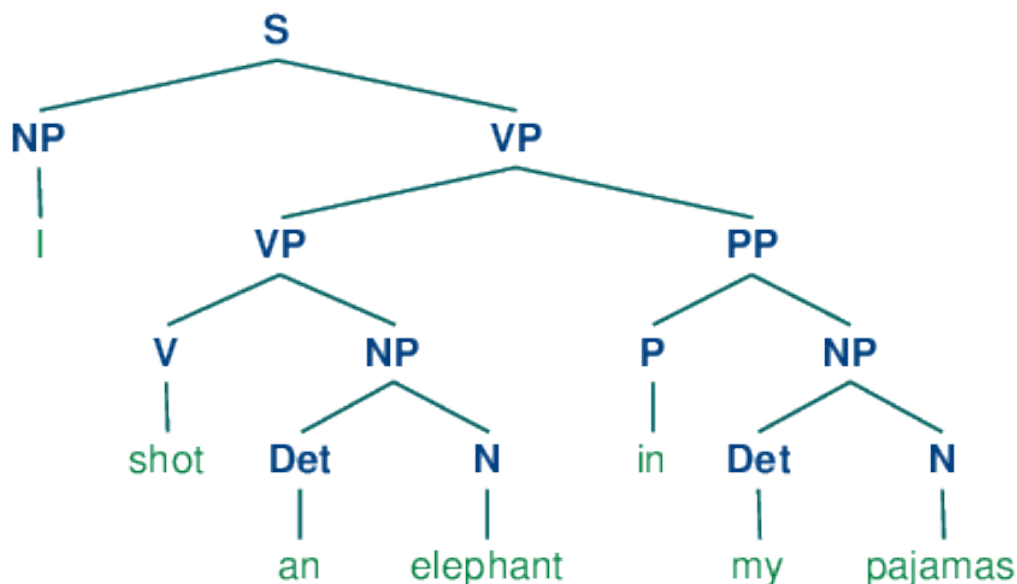
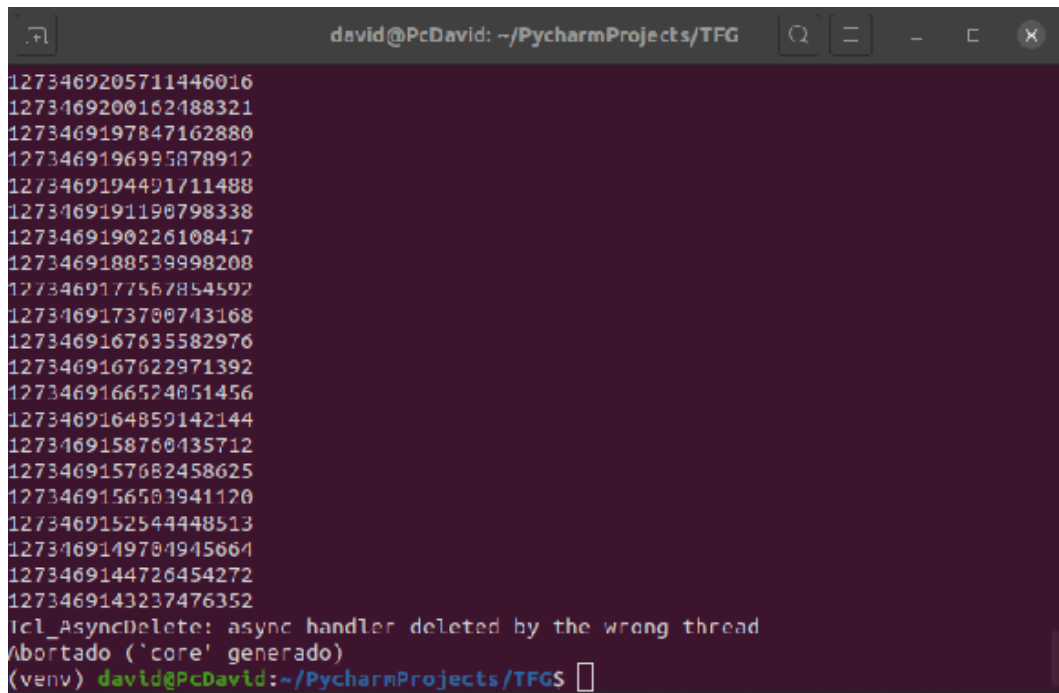


Figura 9.10: Representación de tweet convertido en fotografía

La idea que se intentó implementar es que a medida que se iban creando las estructuras de las fotografías se confeccionara una imagen o un texto estructurado en árbol, para presentárselo al usuario en el análisis final. NLTK tiene unas librerías útiles para la representación gráficamente del árbol con el análisis morfológico, pero las opciones que dan para exportar esa imagen solo están en formato Postscript. Lo más cerca que se ha llegado a implementar ha sido que tras la captura de tweets convertir cada análisis morfosintáctico en formato postscript y este convertirlo en jpg o png. Con esto, se intentó dar solución al problema pero se percibió

otro, la ejecución continuada del marco que crea la imagen en este caso Tkinter produce un fallo en el hilo de ejecución de Python en el sistema.

Se realizaron diferentes pruebas para comprobar que el problema no residía en el ide Pycharm.



```
david@PcDavid: ~/PycharmProjects/TFG
1273469205711446016
1273469200162488321
1273469197847162880
1273469196995878912
1273469194491711488
1273469191190798338
1273469190226108417
1273469188539998208
1273469177567854592
1273469173700743168
1273469167635582976
1273469167622971392
1273469166524051456
1273469164859142144
1273469158760435712
1273469157682458625
1273469156503941120
1273469152544448513
1273469149704945664
1273469144726454272
1273469143237476352
Tcl_AsyncDelete: async handler deleted by the wrong thread
Abortado ('core' generado)
(venv) david@PcDavid: ~/PycharmProjects/TFG
```

Figura 9.11: Error en el hilo de Python al utilizar la librería Tkinter reiteradas veces

En las pruebas que se realizaron el fallo se repetía de forma continuada de dos de cada 6 ejecuciones, debido a esto el análisis morfosintáctico solo se refleja como texto, en la captura de datos cuando se descarga el fichero el usuario.

Como solución a la creación del análisis morfosintáctico se ha utilizado el parser de Stanford para ello se descarga el programa de la su página web[29]. En el paquete viene predefinido el inglés, además hay que descargar el paquete de idioma si no es inglés, se descargó el fichero en español.

Para su uso antes de arrancar la aplicación del TFG, hay que lanzar el servidor, usando un comando en Java para facilidad se ha creado un script en bash para su lanzamiento.

```
#!/bin/sh
cd stanford-corenlp-full-2020-04-20/ && java -mx4g -cp "*"
```

```
edu.stanford.nlp.pipeline.StanfordCoreNLPServer -props
StanfordCoreNLP-spanish.properties -file spanish.txt
-outputFormat json -annotators
“tokenize,ssplit,lemma,pos,ner,depparse” -port 9000
-timeout 30000
```

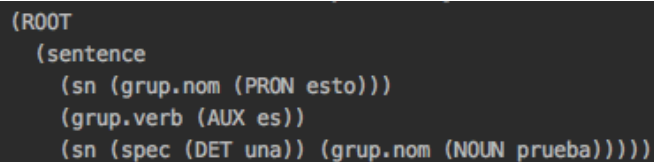
Con esto se lanza un servidor que estará escuchando las peticiones que se pidan desde el TFG. En el código se creará un enlace al servidor arrancado, se tiene que tener en cuenta el puerto y la ip.

```
from pycorenlp import StanfordCoreNLP
nlp = StanfordCoreNLP('http://localhost:9000')
```

En el código se hará una instancia y se realizara las peticiones al servidor.

```
parser = CoreNLPParser()
tw = ("esto es una prueba")
salida = next(parser.raw_parse(tw))
```

La salida del ejemplo `.esto es una prueba .es` la siguiente:



```
(ROOT
  (sentence
    (sn (grup.nom (PRON esto)))
    (grup.verb (AUX es))
    (sn (spec (DET una)) (grup.nom (NOUN prueba)))))
```

Figura 9.12: Ejemplo del CoreNLPParser

Se tiene que convertir la salida a formato string para luego poder pasarlo a el fichero corpus.

9.0.7. Bokeh

Bokeh[30] es una biblioteca para realizar gráficos que son interactivos con el usuario, dando la posibilidad de realizar zoom en el gráfico, realizar un recorte para quedarnos con un subconjunto, herramientas para leyendas inteligentes que realizan un filtrado de los datos representados.

La biblioteca esta en su versión 2.0.1 y está pensada para su uso en navegadores modernos, tiene una gran versatilidad a la hora de diferentes tipos de gráficos desde gráficos de barras, lineales, nubes de partículas.

La elección de esta herramienta fue por parte del tutor del TFG, la finalidad del proyecto es dar análisis en formato html en el que los gráficos estén basados en esta biblioteca para dar al usuario una gran iteración con los datos representados.

Con la realización de muchos experimentos he encontrado problemas al utilizar la librería Bokeh son los siguientes:

- Al representar texto en los ejes x.

Al principio algunos de los gráficos tenían en el eje x el nombre de usuario del que escribió el tweet, esta bien pero hay nombres de usuario que son extremadamente largos por lo que hay descartar los usuarios con nombres muy largos dado que la librería da error con estos nombres.

- Caracteres de emoticonos

Algunos de los usuarios que hay en Twitter tienen el gusto de introducir emoticonos en su nombre, los cuales al representarlos en Bokeh dan error.

- Errores estéticos con los gráficos.

Debido a la cantidad de usuarios que se puede descargar en una captura de datos, se ha tomado la decisión que en el eje x no poner valor.

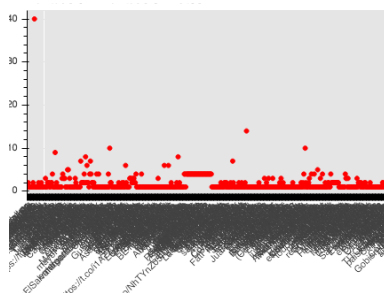


Figura 9.13: Error en el eje x de las graficas de Bokeh

Como se ve en la imagen en nombre de los usuarios al ser tantos se enmaraña creando una mancha negra por esta opción se descartó el eje x para representar información.

Análisis de herramientas de bokeh

Se va a realizar una análisis de cada herramienta, funcionalidad y metáfora que tiene cada icono por separado:

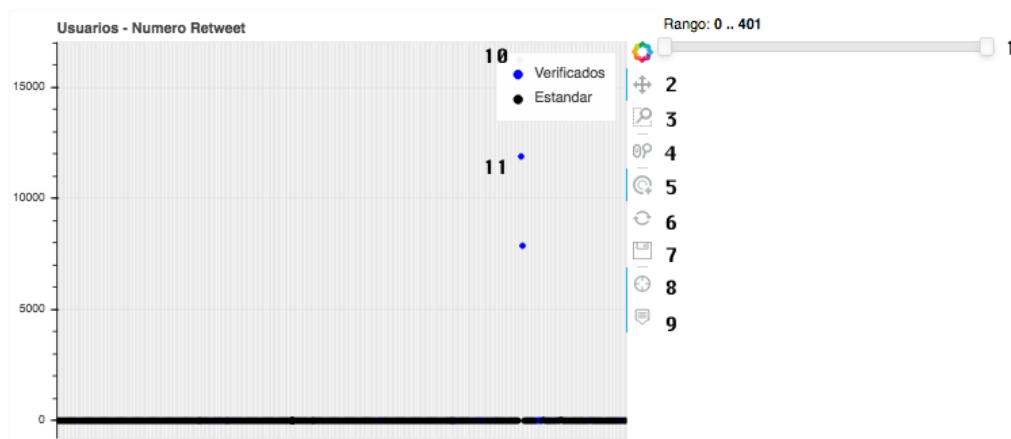


Figura 9.14: Ejemplo de gráfica hecha con Bokeh

1. Slider:

Sus utilidades son ver el número de items que se representan en la gráfica y además sirve para sesgar la captura realizando intervalos en los datos representados. Con mover los dos marcos del Slider se puede ajustar el tamaño del intervalo y donde empieza.

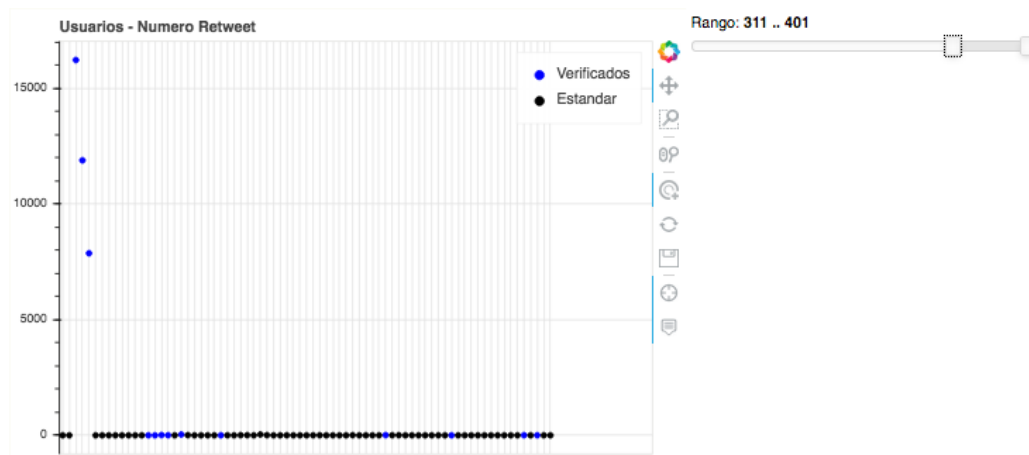


Figura 9.15: Ejemplo de la herramienta Slider

2. Pan:

Su función es la de movernos por el gráfico, pinchando con el ratón y arrastrando nos da la opción de movernos hacia los cuatro sentidos, por esto su metáfora es una aspa con cuatro flechas cada una representadas las direcciones por las que se puede mover.



Figura 9.16: Ejemplo de uso de Pan

3. Box zoom:

El propósito de esta herramienta es realizar un zoom focalizado en un área, la metáfora de este icono es una lupa haciendo referencia al zoom y metido en un cuadrado que

hace referencia al área que hay que realizar para el zoom. Seleccionamos la herramienta, realizamos un área pinchando con el ratón y arrastrando.

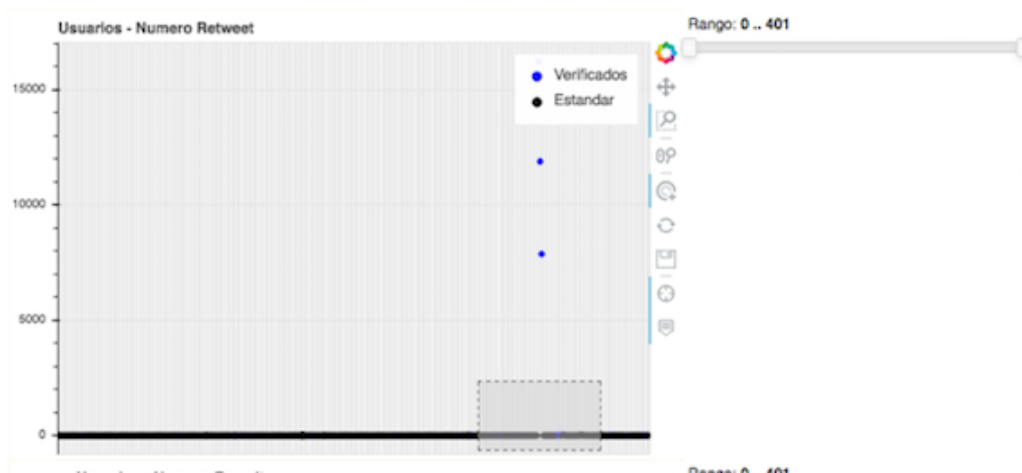


Figura 9.17: Ejemplo de uso del zoom

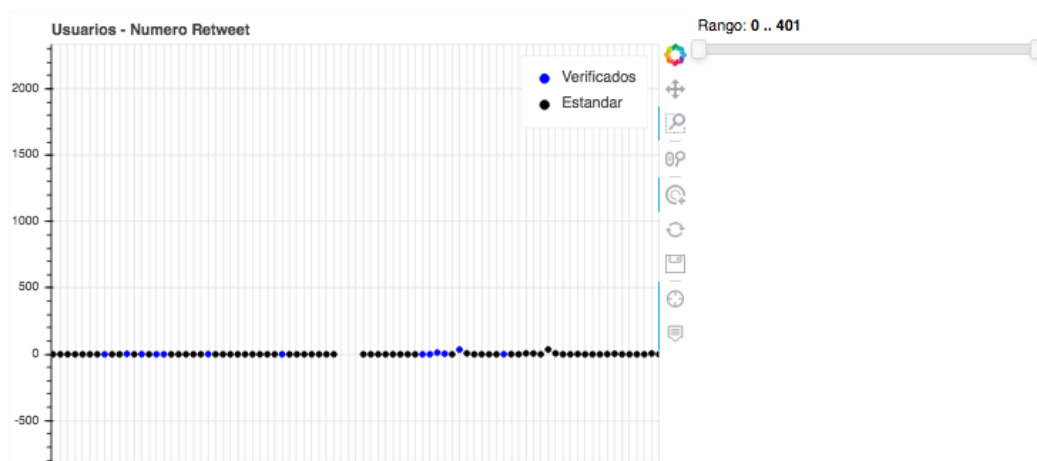


Figura 9.18: Ejemplo de como queda el grafico después de usar zoom

4. Wheel zoom:

Este zoom es con la rueda del ratón, se pone en el cursor en el sitio que queremos

realizar zoom y giramos la rueda del ratón, para utilizarlo como todas las herramientas es primero pinchar primero en la barra para seleccionarla y después colocar el cursor del ratón encima de donde se quiere hacer zoom, se puede realizar el zoom en los sentidos descendente y ascendente. La metáfora del icono es una lupa haciendo referencia al zoom y la rueda del ratón.

5. Tap:

Esta funcionalidad es para desactivar la opción al clicar encima del punto lo que redirección al tweet que representa. Si esta opción está desmarcada al pinchar, no lanzará una nueva pestaña web con el tweet. La metáfora es un círculo haciendo referencia al punto que está representado y el símbolo de más es para representar que tiene una funcionalidad al pinchar. En el caso de este proyecto, la opción que se ha dispuesto en las clicar es la de redirección pero Bokeh provee de más opciones además de abrir una URL.

6. Reset:

El botón reset es para volver al estado por defecto, esto funciona solo en el gráfico, la parte del slider queda bloqueada hasta que el usuario vuelva a mover los cursores del slider.

La metáfora del botón es un símbolo con dos flechas girando sobre si mismas, quiere representar el volver al mismo estado.

7. Save:

Esta funcionalidad es para guardar una captura del gráfico para descargar en formato PNG. La metáfora es un disquete 3.5 muy utilizado en otras aplicaciones que simboliza el guardar un recurso.

8. Crosshair:

Este botón es para activar una ayuda visual en forma de cruz para ajustar mejor el clicar en un punto. La metáfora es un círculo con una mira de un arma hace referencia como a la precisión de un disparo.

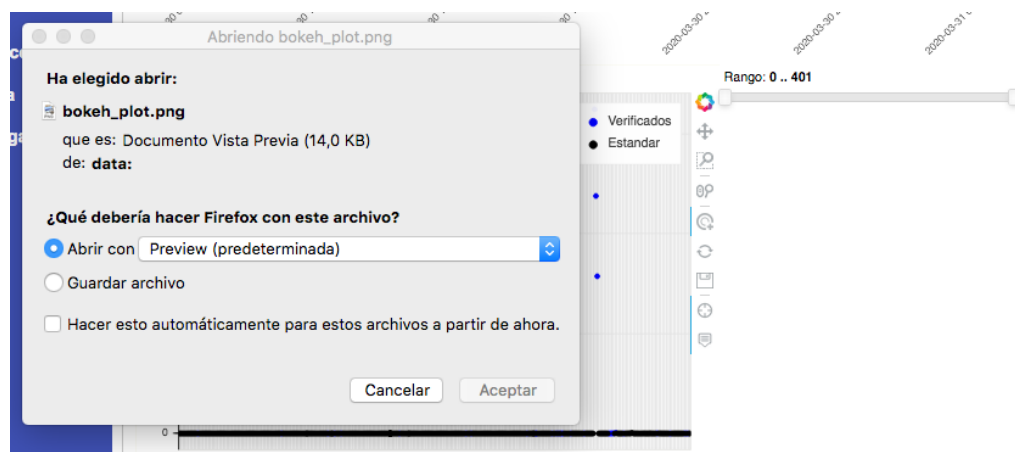


Figura 9.19: Salvar imagen

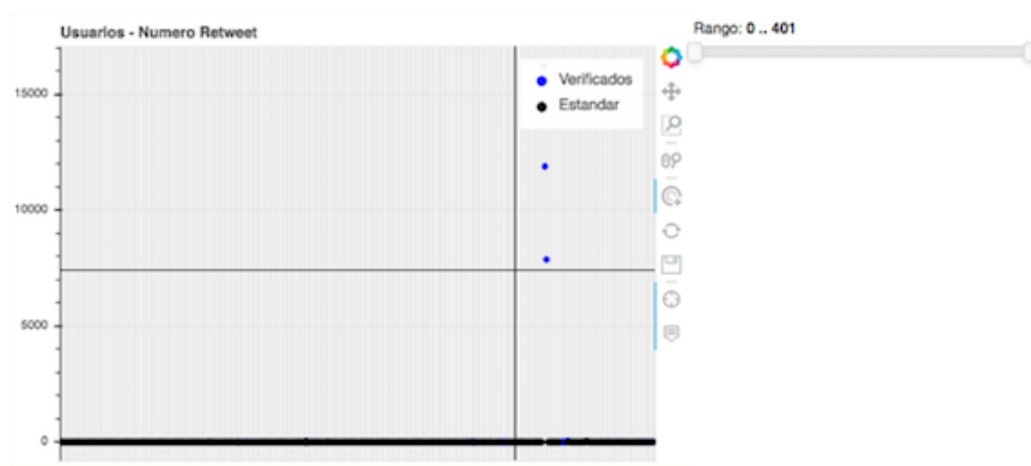


Figura 9.20: Ejemplo del uso de crosshair

9. Hover:

Esta opción es para habilitar la vista rápida sobre los puntos de la gráfica. Si está marcada, mostrará una ventana con los valores, que dependerán de cada gráfica.

10. Leyenda:

El cuadrado que sale a la izquierda tiene los bloques de datos que se le ha introducido a la gráfica, al pinchar en una opción la deshabilita o habilita dependiendo del estado anterior. No todas las gráficas del proyecto tienen leyenda solo en las que hay bloques diferentes de datos.

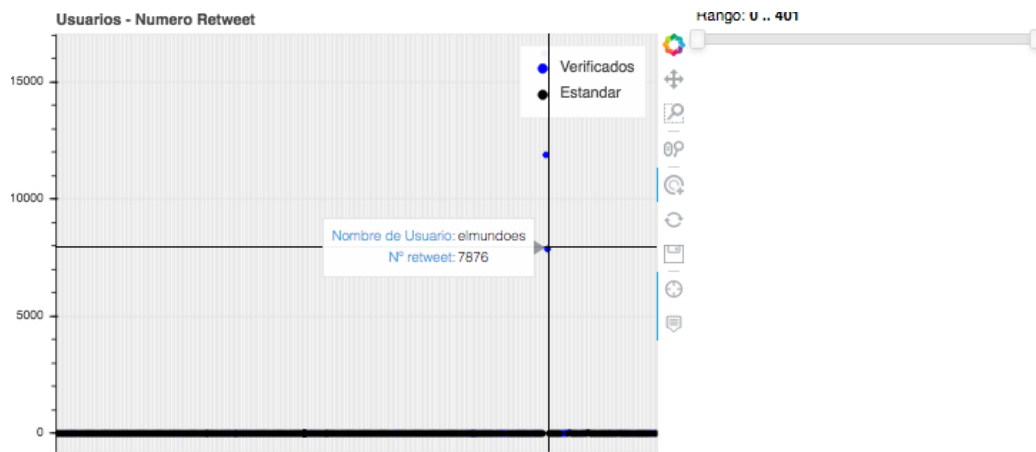


Figura 9.21: Vista rapida sobre los puntos que representan los tweets

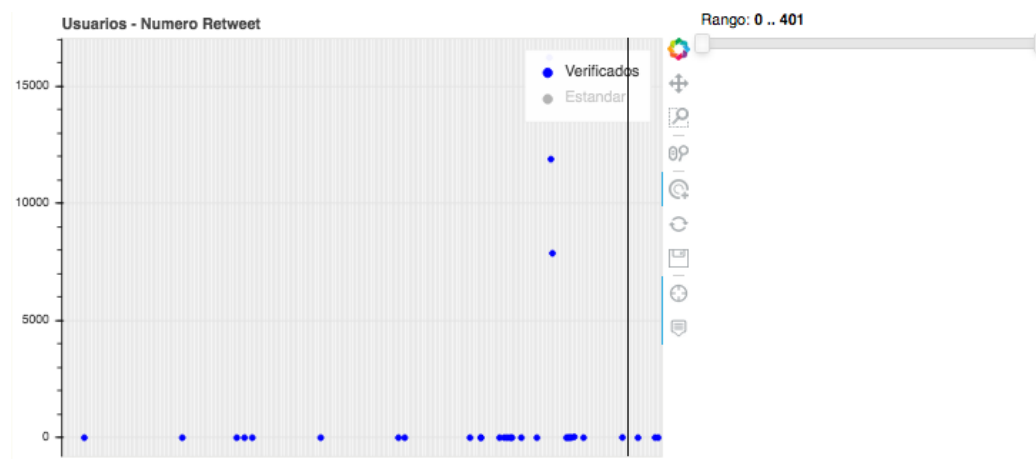


Figura 9.22: Ejemplo de leyenda en Bokeh

En el gráfico de la ilustración se ha pinchado en Estandar para filtrar todos los tweets que no son de usuarios Verificados.

11. Ítem representado:

La representación de los datos en las gráficas, en el caso de este proyecto han sido del tipo de barras o de puntos, pero esta librería tiene más opciones.

Ahora se van a presentar cada uno de los gráficos que se han creado para el proyecto, así como una explicación de su implementación y algunos tratamientos que se han tenido que dar, antes de realizar su representación de los datos.

- Bokeh2: Nombre de usuario y número retweet

Este gráfico relaciona los usuarios de los tweet que en este han tenido más retweets. Este gráfico es para realizar un análisis más social sobre el conjunto de los datos.

Los **valores de entrada** de Bokeh2 la variable que contiene Fichero.csv es el grueso de los datos de la captura.

Valores de salida es el gráfico bokeh será integrado en la página web.

En data contiene el fichero al cual vamos a realizar una consultas, para quedarnos con los usuarios verificados y los normales, hay que crear dos bloques de datos, uno con cada tipo de usuario. Dentro de cada bloque creamos tres lista: la primera con los nombres de usuario, la segunda con el número de retweet que ha tenido cada usuario y por último otra lista con las URL de cada tweet. Con esto creamos una gráfica de puntos en los cuales al pinchar en cada punto nos redirecciona al tweet que representa.

Hay que realizar un tratamiento para el nombre de los usuarios, por uno de los problemas que tiene Bokeh el tamaño de texto que se puede representar, por ello hay que ver cada nombre y desechar los usuarios que sean muy largos.

Después de estar preparados, los dos bloques de datos en nuestra función, source y source2, creamos nuestra figura a la cual le damos como parámetro x el rango de valores de usuario que tenemos. El parámetro tools son todas las herramientas que va a tener disponible nuestra gráfica, en bokeh han sido antes descritas, por último un título para la gráfica.

Sobre la figura creada aplicamos la inserción de los puntos:

```
p.circle('x', 'y', color="yellow", size=5,  
source=source2, alpha=1.5, legend_label="Verificados")
```

```
p.circle('x', 'y', color="black", size=5,  
source=source, alpha=1.5, legend_label="Estandar")
```

Los puntos serán de dos colores dependiendo del bloque de datos que inserten en el caso de los usuarios verificados, el color del punto será azul y el caso de los usuarios estándar será de color negro. Una cosa importante es el parámetro legend_label el cual genera una categoría en la leyenda, esto ayuda a la hora de filtrar el tipo de usuarios pinchando en la leyenda.

Para dar más interacción con el gráfico se utiliza, función callback de bokeh, a la cual le asignamos la función OpenUrl y la lista de url de cada tweet, así podemos redireccionar al tweet pinchando en el punto.

Aunque se puede realizar zoom sobre el gráfico de los datos, se le puede añadir una vista rápida al colocar el ratón sobre el punto. La función hovertool nos da la posibilidad de programar que valores puede ver el usuario al colocar el ratón encima del punto representado en la gráfica.

Por último y con doble funcionalidad, se ha colocado un slider es una barra diseñada en Javascript, sus usos son visualizar un valor con el número de puntos representados y aplicar un sesgo entre los datos.

Ejemplo: el usuario ve que hay 25933 tweets, pero hoy va a realizar un estudio sobre los 100 primeros.

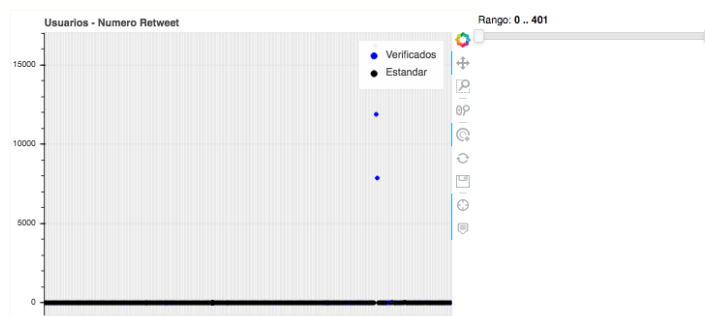


Figura 9.23: Ejemplo de todos los tweets

Moviendo el cursor podemos sesgar los valores entre un rango.

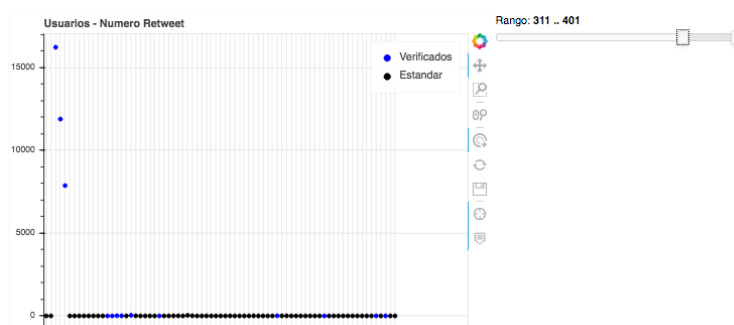


Figura 9.24: Ejemplo de uso del slider

```
def bokeh2(archivo):

    data = pd.read_csv(archivo, delimiter='\t',
engine="python",
quotechar='"', error_bad_lines=False)    #Perfecto
output_file('filename.html')
x = data['NombreUsuario'].unique().tolist()
Estandar = data[data['userVerified'] == False]

    listaA = Estandar['NombreUsuario'].tolist()
    listaB = Estandar['retweetCount'].tolist()
    ListaC = Estandar['URLTweet'].tolist()
    listaFinalA = list()
    listaFinalB = list()
    listaFinalC = list()

    contador = 0
    for a ,b in zip(listaA,listaB):

        if len(a) < 30:

            listaFinalA.append(a)
            listaFinalB.append(b)
            listaFinalC.append(ListaC[contador])

        contador+=1
    output_file("openurl.html")
    source = ColumnDataSource(data=dict(
        x=listaFinalA,
        y=listaFinalB,
        url=listaFinalC
```

```
))
verificados = data[data['userVerified'] == True]

listaA2 = verificados['NombreUsuario'].tolist()
listaB2 = verificados['retweetCount'].tolist()
ListaC2 = verificados['URLTweet'].tolist()
listaFinalA2 = list()
listaFinalB2 = list()
listaFinalC2 = list()

contador2 = 0
for a, b in zip(listaA2, listaB2):

    if len(a) < 30:
        listaFinalA2.append(a)
        listaFinalB2.append(b)
        listaFinalC2.append(ListaC2[contador2])

    contador2 += 1

source2 = ColumnDataSource(data=dict(
    x=listaFinalA2,
    y=listaFinalB2,
    url=listaFinalC2

))

p = figure(x_range=x, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap,
           wheel_zoom", title="Usuarios - Numero Retweet")
p.xaxis.visible=False

p.circle('x', 'y', color="yellow", size=5,
         source=source2, alpha=1.5, legend_label="Verificados")
p.circle('x', 'y', color="black", size=5,
         source=source, alpha=1.5, legend_label="Estandar")
p.xaxis.major_label_orientation = pi / 4
p.background = "beige"
```



```

url = "@url"
taptool = p.select(type=TapTool)
taptool.callback = OpenURL(url=url)

hover = HoverTool(tooltips=[("Nombre de Usuario", "@x"), ("Nº retweet", "@y")])
p.add_tools(hover)

p.legend.click_policy = "hide"

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
""")

range_slider = RangeSlider(start=0, end=len(data), value=(0, len(data)), step=1)
range_slider.js_on_change('value', callback)

layout = row(p, widgetbox(range_slider))
# show(layout)

return layout

```

- Bokeh3: Usuarios - Número Favoritos

Es una gráfica en la que se representa la relación de los tweets y el número de favoritos que ha tenido, la implementación es muy parecida a bokeh2, la diferencia es que el conjunto de datos se selecciona el campo de favoritos de cada tweet.

Valores entrada el conjunto de datos capturados y los **valores de salida** del gráfico que se añadirán a la página que ve de resultado a el usuario.

```

def bokeh3(archivo):
    data = pd.read_csv(archivo, delimiter='\t',
                      engine="python",
                      quotechar='\"', error_bad_lines=False)
    output_file('filename.html')

```

```
x = data['NombreUsuario'].unique().tolist()
color = data['URLTweet'].unique().tolist()
Estandar = data[data['userVerified'] == False]

output_file("openurl.html")
source = ColumnDataSource(data=dict(
    x=Estandar['NombreUsuario'].tolist(),
    y=Estandar['FavoriteCount'].tolist(),
    url=Estandar['URLTweet'].tolist()

))
verificados = data[data['userVerified'] == True]
source2 = ColumnDataSource(data=dict(
    x=verificados['NombreUsuario'].tolist(),
    y=verificados['FavoriteCount'].tolist(),
    url=verificados['URLTweet'].tolist()

))

p = figure(x_range=x, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap,
           wheel_zoom",
           title="Usuarios - Numero Favoritos")
p.xaxis.visible=False

p.circle('x', 'y', color="blue", size=5, source=source2, alpha=1.5,
        legend="Verificados")
p.circle('x', 'y', color="black", size=5, source=source, alpha=1.5,
        legend="Estandar")
p.xaxis.major_label_orientation = pi / 4

p.background = "beige"
url = "@url"
taptool = p.select(type=TapTool)
taptool.callback = OpenURL(url=url)

p.legend.click_policy = "hide"
```

```

hover = HoverTool(tooltips=[("Nombre de Usuario", "@x")
, ("Nº favoritos", "@y")])
p.add_tools(hover)

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
    """)

range_slider = RangeSlider(start=0, end=len(data)
, value=(0,
len(data)), step=1, title="Rango")
range_slider.js_on_change('value', callback)

layout = row(p, widgetbox(range_slider))
# show(layout)

return layout

```

- Bokeh4: número de tweets por tiempo

Este gráfico muestra el número de tweets que han sido escritos durante la captura, así podemos ver el periodo en el cual los usuarios han escrito más sobre el tema de búsqueda y ver como las franjas temporales en las cuales hay más tweets que se han escrito.

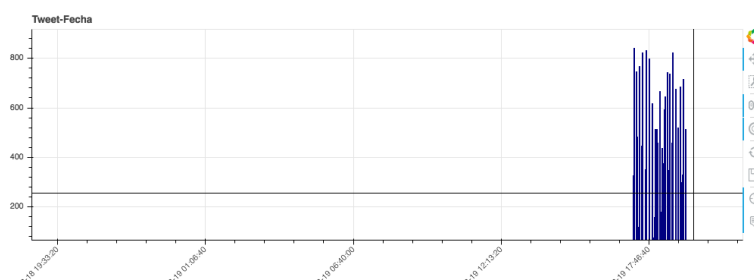


Figura 9.25: Ejemplo de bokeh en tweets por tiempo

Si con las herramientas antes nombradas se hace zoom, se puede precisar en que momento se han escrito los tweets.

Para realizar este gráfico, hay un tratamiento anterior de los tweets para así mostrar los datos en el gráfico. El inconveniente es que hay que cambiar el formato de las fechas capturadas en Twitter, hay que adaptarlas al tipo de dato para que la librería Bokeh pueda representarla.

Lo primero es quitar la parte que representa los segundos, la razón es para que la gráfica represente por minutos, así las líneas de tweet están más condensadas para apreciar mejor el número de tweet por instante de tiempo.

```
data['fecha'] = data['fecha'].apply(lambda t: t.replace(second=0))
```

Después de esto creamos una lista con las fechas que se van a representar.

```
fecha = data.sort_values(by='fecha')
h = fecha['fecha'].tolist()
lista = list()
for hi in h:
    lista.append(h.count(hi))
```

Más tarde, creamos nuestra figura a la cual le pasamos como parámetros el título de la gráfica y las herramientas de la gráfica.

Ahora introducimos las barras en la gráfica y como fuente de datos la lista con las fechas antes capturadas.

```
p.vbar(
    x=h,
    width=0.9,
    bottom=0,
    top=lista,
    color='navy'
)
```

Por último y más importante, hay que representar los valores en la gráfica introduciendo el formato de fechas que se van a representar. Esto es lo más complicado de esta gráfica, el como convertir las fechas de Twitter en fechas que reconozca Bokeh.

```

p.xaxis.formatter = DatetimeTickFormatter(
microseconds=['%Y-%m-%d %H:%M:%S.%f'],
milliseconds=['%Y-%m-%d %H:%M:%S.%3N'],
seconds=["%Y-%m-%d %H:%M:%S"],
minsec=["%Y-%m-%d %H:%M:%S"],
minutes=["%Y-%m-%d %H:%M:%S"],
hourmin=["%Y-%m-%d %H:%M:%S"],
hours=["%Y-%m-%d %H:%M:%S"],
days=["%Y-%m-%d %H:%M:%S"],
months=["%Y-%m-%d %H:%M:%S"],
years=["%Y-%m-%d %H:%M:%S"],)

```

- Bokeh5: Puntuación datos de wikipedia

En la gráfica siguiente, se presenta las puntuaciones de los tweets respecto a los datos obtenidos de Wikipedia. Cada punto representado en el eje x es un tweet y la altura que tiene es la puntuación, esta son las palabras encontradas dentro del tweet que están en el conjunto de Wikipedia.

En este caso para diferenciar gráficamente cada tweet, se ha tomado el id del tweet que es un campo único que esta en el conjunto de captura de datos. Los otros valores que se toman del fichero de entrada son el texto del tweet, el cual se introduce en la vista rápida del gráfico junto con la puntuación del tweet, además de la url de cada tweet para así en el momento de pinchar en el punto nos redirección al tweet.

Esta gráfica no tiene tratamiento de la información, se cogen directamente los datos y se plasman en ella. **Valores de entrada** el fichero PuntuacionWikipedia.csv y los **Valores de salida** el grafico.

- Bokeh6:Puntuación de datos de los valores introducidos

En esta gráfica, se genera por medio de los datos que ha creado el análisis de los tweet y cotejarlos con la entrada de datos del usuario.

Valores de entrada es el fichero PuntuaciónFichero.csv y en este fichero lo cargamos en un dataset de pandas, después realizamos una consulta sobre el dataset y nos quedamos con las columnas: 'id' que será el valor único para separar cada punto representado. Este id es el id de cada tweet, la "puntuación"del tweet que en este caso es el número de



Figura 9.26: Ejemplo de bokeh de wikipedia

veces que ha salido algunas de las palabras introducidas por el usuario en los tweets capturados.

Luego en el hover, que es la vista rápida que se coloca en cada punto de la gráfica, se ubica la puntuación y el texto del tweet, además tiene la opción que al pinchar en el punto redirrecciona al tweet escrito en la plataforma Twitter.

Los **valores de salida** es la creación del gráfico

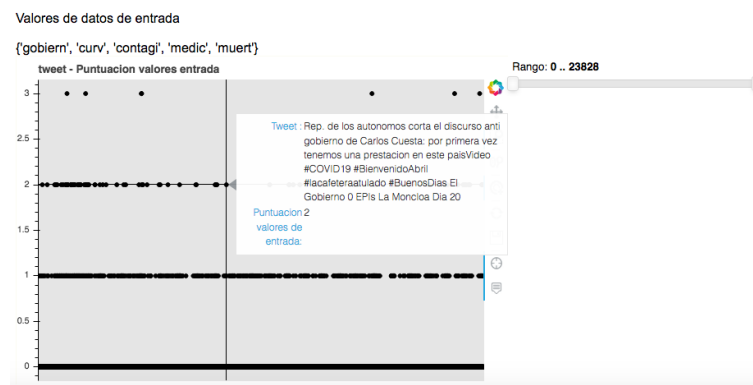


Figura 9.27: Ejemplo de gráfica de valores con Bokeh

```
def bokeh6(archivo):
```

```
    data = pd.read_csv(archivo, delimiter='\\t', engine="python")
    output_file('filename.html')
    x = data['id'].unique().tolist()
```

```

y = data['puntuacion'].unique().tolist()

color = data['URL'].unique().tolist()

output_file("openurl.html")
source = ColumnDataSource(data=dict(
    x=data['id'].tolist(),
    y=data['puntuacion'].tolist(),
    url=data['URL'].tolist(),
    texto = data['texto'].tolist()

))

p = figure(x_range=x, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap,

           wheel_zoom"
           ,
           title="tweet - Puntuacion valores entrada ")
p.xaxis.visible = False

p.circle('x', 'y', color="black", size=5, source=source, alpha=1.5)

p.xaxis.major_label_orientation = pi / 4
p.background = "beige"
url = "@url"
taptool = p.select(type=TapTool)
taptool.callback = OpenURL(url=url)

hover = HoverTool(tooltips=[("Tweet  ", "@texto")
, ("Puntuacion valores de entrada", "@y")])
p.add_tools(hover)

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
    """)

```

```

range_slider = RangeSlider(start=0, end=
len(data), value=(0, len(data)), step=1, title="Rango")
range_slider.js_on_change('value', callback)

layout = row(p, widgetbox(range_slider))
# show(layout)

return layout

```

- Bokeh9:

En el gráfico siguiente se va a representar las veces nombradas a algún usuario, **valores entrada** el fichero Usuarios.csv y como **valores salida** el gráfico.

Realizamos una consulta y creamos dos listas con los valores de 'Usuarios' y 'número', en este caso, sí tenemos que realizar un tratamiento en los datos, ya que puede que haya un usuario que tengan un nombre muy largo y Bokeh no sea capaz de representarlo.

Como en este gráfico no tenemos la URL de los tweet dado que son usuarios, al clicar en el punto nos irá al usuario que representa cada punto. Se pensó en poner los usuarios diferenciados en estándar y verificados, pero en este caso los usuarios que tenemos no son devueltos en la clase status que devuelve la Api de Twitter. Por ello, son captados del texto por tener delante el símbolo "@" que indica ser un usuario. Si se hubiera tenido una claves de Api de Twitter de una categoría superior, se podría realizar un estudio y posteriormente una representación de los usuarios verificados nombrados y los normales, pero en el caso de las claves que tenemos en este proyecto, al realizar todas las consultas una por usuario nombrado daría los siguientes inconvenientes:

- Principal el gasto de recursos de conexión a la Api.
- Aumentaría el tiempo de espera cada vez que se supere el limite de uso de la Api y el análisis de los datos se alargue.

```
def bokeh9(archivo):
```

```

    data = pd.read_csv(archivo, delimiter='\t', engine="python")

    listaA = data['Usuario'].tolist()
    listaB = data['numero'].tolist()

```



```
listaFinalA = list()
listaFinalB = list()

for a ,b in zip(listaA,listaB):
    if len(a) < 30:
        valor = a.find("@")

        if valor != -1:

            a = a[valor:]

        listaFinalA.append(a)
        listaFinalB.append(b)

output_file('filename.html')

output_file("openurl.html")
source = ColumnDataSource(data=dict(
    x=listaFinalA,
    y=listaFinalB
))
p = figure(x_range=listaFinalA, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap,wheel_zoom",
           title="Puntuación - Usuarios ")
p.xaxis.visible=False

p.circle('x', 'y', color="black", size=5, source=source, alpha=1.5)

p.xaxis.major_label_orientation = pi / 4
p.background = "beige"

url = "http://www.twitter.com/@x"
taptool = p.select(type=TapTool)
taptool.callback = OpenURL(url=url)

hover = HoverTool(tooltips=[("Id tweet  ", "@x"),
```

```

("Numero de usuarios nombrados", "@y"))
p.add_tools(hover)

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
    """)

range_slider = RangeSlider(start=0, end=len(data)
, value=(0, len(data)), step=1, title="Rango")
range_slider.js_on_change('value', callback)

layout = row(p, widgetbox(range_slider))

return layout

```

- Bokeh11: Gráfica de Agrupamiento

En este gráfico nos hemos apoyado en unas estructuras para realizarlo, se pensó en implementar una forma de representar los cluster del agrupamiento de tal manera que gráficamente se pudiera separar. Además de que antes de representar los datos en la gráfica también mostrará las palabras más utilizadas en cada cluster.

Para ello la estructura que se ha utilizado para apoyarse, ha sido una lista de colores para diferenciar cada tipo de cluster.

En muchas capturas de datos, no ha habido datos suficientes para realizar el análisis por lo que habría que representar una gráfica vacía si el contenido de **valores de entrada** es 0 se crea una gráfica vacía. Después de tener el mínimo número de valores para realizar la gráfica, tenemos que encontrar el número de tipo de cluster que hay. Para esto, realizamos una consulta al dataset y nos quedamos con el número único, esto nos devuelve una lista con los números de forma única sin repetida.

```
colores = data['cluster'].unique().tolist()
```

En colores estarán el número de tipos de cluster que hay en el conjunto, seguido tenemos que realizar un while para realizar una consulta de cada tipo de cluster y colocarlos en el gráfico con un color diferente.

```
while a < len(colores):
    cluster = data[data['cluster'] == a]
    source = ColumnDataSource(data=dict(
        x=cluster['id'].tolist(),
        y=cluster['cluster'].tolist(),
        url=cluster['URL'].tolist()
    ))
    p.circle('x', 'y', color=lColores[a], size=5,
            source=source, alpha=1.5, legend="cluster " +str( a))
    a += 1
```

Los valores con los que nos quedamos para realizar el gráfico son id, cluster, URL. Así el usuario si le interesa algún tipo de cluster puede filtrarlos con la leyenda y si quiere profundizar en el tweet al pinchar lo redirección al tweet en Twitter.

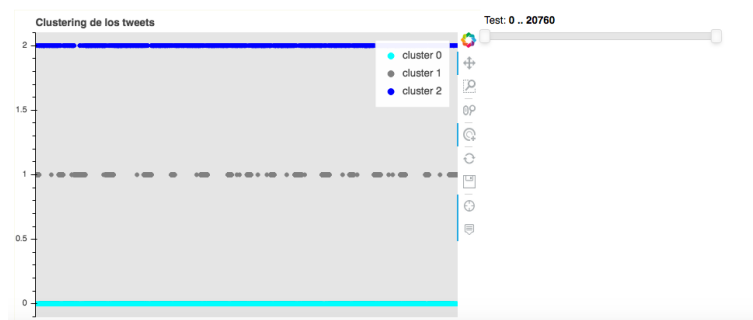


Figura 9.28: Ejemplo de gráfica de agrupamiento con Bokeh

En la gráfica aparece tres cluster distintos, están representados con tres colores diferentes, en el caso de que solo nos interesará alguno de ellos al pinchar en la leyenda descartando cluster.


```

        legend="cluster " +str( a))
        a += 1
    p.xaxis.major_label_orientation = pi / 4
    p.background = "beige"
    url = "@url"
    taptool = p.select(type=TapTool)
    taptool.callback = OpenURL(url=url)

    hover = HoverTool(tooltips=[("Id tweet  ", "@x"),
                                ("Clustering de los tweets", "@y")])
    p.add_tools(hover)
    p.legend.click_policy = "hide"

    callback = CustomJS(args=dict(p=p), code="""
        var a = cb_obj.value;
        p.x_range.start = a[0];
        p.x_range.end = a[1];
    """)

    range_slider = RangeSlider(start=0,
                                end=len(data), value=(0, len(data)), step=1, title="Rango")
    range_slider.js_on_change('value', callback)

    layout = row(p, widgetbox(range_slider))

    return layout
else:
    p = figure( plot_width=600, plot_height=400
    ,
                tools="pan,crosshair,reset,save,
                box_zoom,tap,
                wheel_zoom", title="Clustering de los tweets")

    callback = CustomJS(args=dict(p=p), code="""
        var a = cb_obj.value;
        p.x_range.start = a[0];
        p.x_range.end = a[1];
    """)

```

```

        """)

    range_slider = RangeSlider(start=0, end=1, value=(0, 1), step=1,
                               title="Rango")
    range_slider.js_on_change('value', callback)

    layout = row(p, widgetbox(range_slider))
    return layout

```

- Bokeh12: Gráfica de Hashtag En este gráfico se va a analizar los hashtag más utilizados en la captura de datos. Aquí hay que realizar un tratamiento a la información debido a que los hashtag que utilizan los usuarios de Twitter muchas veces son muy largos y Bokeh no es capaz de representarlos y además, hay que recortar algunos: Ejemplo: culpa del #Coronavirus

En este caso tenemos que realizar un recorte en el tweet para quedarnos con coronavirus.

```

valor = a.find("#")
if valor != -1:
    a = a[valor:]

```

Después de realizar el recorte sobre los hashtag cargamos los datos en el gráfico y retornamos el gráfico.

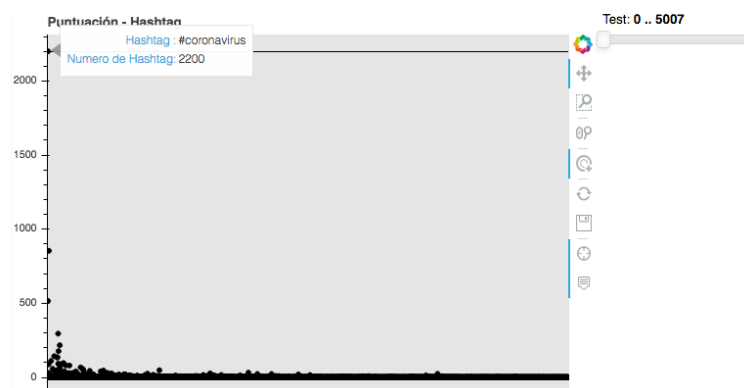


Figura 9.30: Gráfica de la captura de hashtag

```
def bokeh12(archivo):

    data = pd.read_csv(archivo, delimiter='\t', engine="python")

    listaA = data['Hashtag'].tolist()
    listaB = data['numero'].tolist()
    listaFinalA = list()
    listaFinalB = list()

    for a ,b in zip(listaA,listaB):
        if len(a) < 30:
            valor = a.find("#")

            if valor != -1:

                a = a[valor:]

            listaFinalA.append(a)
            listaFinalB.append(b)

    output_file('filename.html')

    output_file("openurl.html")
    source = ColumnDataSource(data=dict(
        x=listaFinalA,
        y=listaFinalB
```

```

))

p = figure(x_range=listaFinalA, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap,wheel_zoom",
           title="Puntuación - Hashtag ")
p.xaxis.visible = False

p.circle('x', 'y', color="black", size=5, source=source, alpha=1.5)

p.xaxis.major_label_orientation = pi / 4
p.background = "beige"

taptool = p.select(type=TapTool)

hover = HoverTool(tooltips=[("Hashtag  ",
                             "@x"), ("Numero de Hashtag", "@y")])
p.add_tools(hover)

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
    """)

range_slider = RangeSlider(start=0, end=len(data)
                           , value=(0, len(data)), step=1, title="Rango")
range_slider.js_on_change('value', callback)

layout = row(p, widgetbox(range_slider))
return layout

```

- Bokeh13: Gráfica de localidades **Los valores de entrada** son los datos de Localidades.csv, de este archivo se van a utilizar los campos: localidad y el número

de veces que se ha repetido.

Observaciones:

- Algunas de las palabras, las toma como localidades sin serlo, esto es por problemas de Spacy. Muchas veces, palabras como hospital o COVID la ha tomado como lugares de manera errónea.

Hay que realizar un tratamiento de la información antes de presentarla por el tamaño de las palabras para que pueda representarla Bokeh. El resto de la función hace los mismos pasos que las anteriores, crea el conjunto de datos y lo introduce en la gráfica en forma de puntos, esta gráfica no tiene ninguna acción predeterminada al pinchar en un punto.

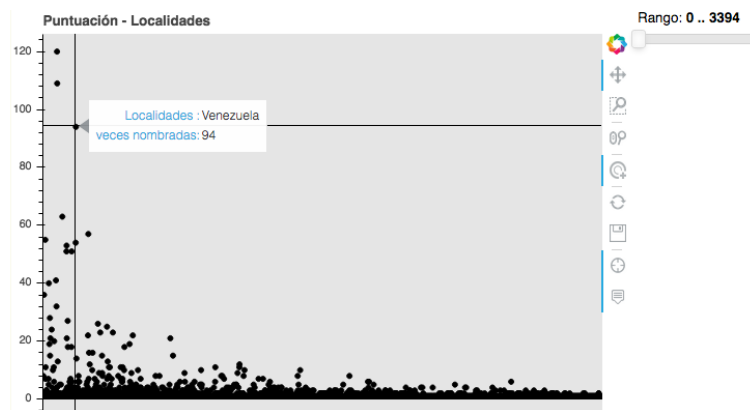


Figura 9.31: Ejemplo de gráfica de localidades

```
def bokeh13(archivo):
```

```
    data = pd.read_csv(archivo, delimiter='\\t', engine="python")
```

```
    listaA = data['localidad'].tolist()
```

```
    listaB = data['numero'].tolist()
```

```
    listaFinalA = list()
```

```
    listaFinalB = list()
```

```
    for a ,b in zip(listaA,listaB):
```

```

        if len(a) < 30:

            listaFinalA.append(a)
            listaFinalB.append(b)

output_file('filename.html')
output_file("openurl.html")
source = ColumnDataSource(data=dict(
    x=listaFinalA,
    y=listaFinalB

))

p = figure(x_range=listaFinalA, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap,wheel_zoom",
           title="Puntuación - Localidades ")
p.xaxis.visible= False

p.circle('x', 'y', color="black", size=5, source=source, alpha=1.5)

p.xaxis.major_label_orientation = pi / 4
p.background = "beige"

hover = HoverTool(tooltips=[("Localidades  ", "@x"), ("veces nombradas",
"@y")])
p.add_tools(hover)

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
    """)
range_slider = RangeSlider(start=0, end=len(data)

, value=(0, len(data)), step=1, title="Rango")
range_slider.js_on_change('value', callback)
layout = row(p, widgetbox(range_slider))

```

```
return layout
```

- Bokeh14: Gráfica de organizaciones En el gráfico de organizaciones se realiza una ponderación de las organizaciones nombradas en los tweet, con esto ayudamos al usuario a realizar un análisis de las organizaciones más nombradas sobre un tema de búsqueda. En este gráfico solo se ha tenido que tener en cuenta la longitud de las palabras a la hora de representarlas y que sean palabras vacías, el resto de la función es igual a sus antecesoras. Este gráfico no tiene ninguna funcionalidad al pinchar sobre uno de los puntos, esto es debido a que si se realizará una búsqueda en Twitter de cada organización para ver si tiene cuenta de Twitter, sería lento, además de gastar el recurso del límite de uso de la Api.

Observaciones:

- Muchas veces coloca como organización a palabras que no lo son, ejemplo: covid lo reconoce Spacy como una organización.

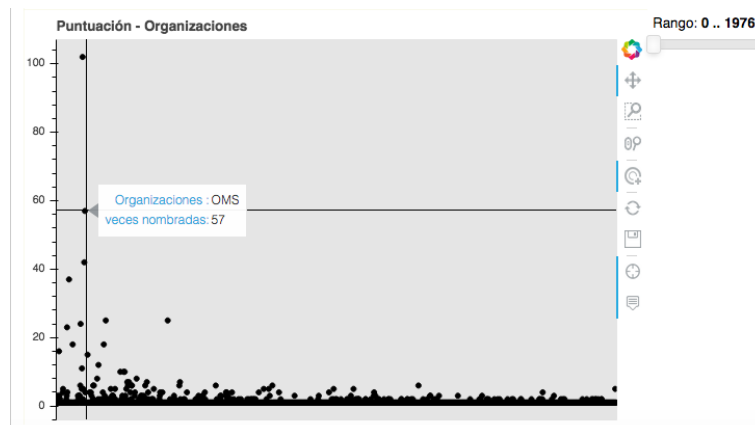


Figura 9.32: Ejemplo de gráfica de localidades

```
def bokeh14(archivo):

    data = pd.read_csv(archivo, delimiter='\\t', engine="python")

    listaA = data['Organizaciones'].tolist()
```

```

listaB = data['numero'].tolist()
listaFinalA = list()
listaFinalB = list()
for a ,b in zip(listaA,listaB):
    a = str(a)
    if a != "" and a.isnumeric() == False:
        if len(a) < 30:

            listaFinalA.append(a)
            listaFinalB.append(b)
output_file('filename.html')
output_file("openurl.html")
source = ColumnDataSource(data=dict(
    x=listaFinalA,
    y=listaFinalB
))
p = figure(x_range=listaFinalA, plot_width=600, plot_height=400,
           tools="pan,crosshair,reset,save,box_zoom,tap
           ,wheel_zoom", title="Puntuación - Organizaciones ")
p.xaxis.visible= False
p.circle('x', 'y', color="black", size=5, source=source, alpha=1.5)
p.xaxis.major_label_orientation = pi / 4
p.background = "beige"
hover = HoverTool(tooltips=[
    ("Organizaciones  ", "@x"), ("veces nombradas", "@y")])
p.add_tools(hover)

callback = CustomJS(args=dict(p=p), code="""
    var a = cb_obj.value;
    p.x_range.start = a[0];
    p.x_range.end = a[1];
    """)
range_slider = RangeSlider(start=0, end=len(data), value=(0, len(data))
, step=1, title="Rango")
range_slider.js_on_change('value', callback)
layout = row(p, widgetbox(range_slider))
return layout

```

9.0.8. Clases creadas de apoyo

Se va a presentar las clases de apoyo para realizar el análisis de la captura de tweet.

- Fichero nodo Dentro del fichero nodo, está compuesta por dos clases nodo1 y nodo4, cada uno es para realizar un tipo de captura de datos.
- Nodo1 Nodo1 es una clase para realizar el conteo de una palabra, el uso por ejemplo es para la obtención de los análisis de localidades, organizaciones y usuarios. A la hora de utilizarla, creamos un diccionario que actuará como un árbol binario, el problema es que hay que implementar el nodo que contiene dicho árbol. Este nodo tiene como valor clave la palabra que representa y el objeto que alberga es de tipo nodo1, dentro de la clase nodo1, está compuesta por la palabra y el contador del número de veces que aparece. La función sumar es para aumentar en uno el contador.

```
class nodo1():
    palabra = ''
    contador = 1
    def __init__(self,p):
        self.palabra = p
        self.contador = 1

    def sumar(self):
        self.contador += 1
    def palabra(self):
        return str(self.palabra)
```

- Nodo4 Esta clase es para realizar los análisis de los diccionarios con la Api de Wikipedia y con los valores escritos por el usuario. Para su uso creamos un diccionario y los nodos dentro del diccionario serán del tipo nodo4. Los componentes de la clase nodo4 son los siguientes: el id del tweet, texto del tweet, la puntuación que se irá incrementando a medida que se encuentren palabras en el tweet que estén contenidas en el diccionario creado con la Api de Wikipedia y con los valores introducidos con el usuario, la URL del tweet para que luego al pinchar se pueda ir al tweet y el usuario que lo ha escrito, la lista es para la hora de comparar.

```
class nodo4():
    id = ""
    texto = ""
    puntuacion = 0
    URL = ""
    lista = ""
    usuario = ""
    def __init__(self,id,t,p,u,l,us):
        self.id = id
        self.texto = t
        self.puntuacion = p
        self.URL = u
        self.lista
        self.usuario = us

    def get_texto(self):
        return str(self.texto)
    def get_URL(self):
        return str(self.URL)
    def get_puntuacion(self):
        return str(self.puntuacion)
    def get_lista(self):
        return self.lista
    def get_usuario(self):
        return self.usuario
    def get_id(self):
        return self.id
```

- **Fichero clusterInfo** Esta clase es la que devuelve la aplicaciones que se encargan de realizar el agrupamiento y formar los cluster. La razón de su uso se explica más adelante, su problema que es la caché del servidor de flask. Este servidor crea una caché que se queda en los navegadores, entonces al realizar la petición a la página web que va albergar el proyecto. Las primeras veces los gráficos que se creaban para el método del codo siempre devolvía el mismo gráfico. Para solucionar este problema se guarda la información en una variable y se retorna para luego mostrarla. De esta forma siempre sale un gráfico distinto, además otro de los usos es saber el número de tweets que se han utilizado para el entrenamiento y cuantos para la prueba con Kmeans. La clase tiene otra funcionalidad muy importante, se pensó en colocar en la página web las palabras más representativas de cada cluster. Por ello, uno de los atributos de la clase clusterinfo es una lista con estas palabras, los otros valores son enteros con el número de tweets de entrenamiento y de prueba.

```
clusterinfo():
    plot = plt
    completo = 0
    entrenamiento = 0
    prueba = 0
    clusters = list()
    def __init__(self,p,c,e,pr,cl):
        self.plot=p
        self.completo =c
        self.entrenamiento = e
        self.prueba=pr
        self.clusters = cl

    def get_plot(self):
        return self.plot
    def get_completo(self):
        return self.completo
    def get_prueba(self):
        return self.prueba
    def mostrar(self):
        self.plot.show()
    def get_cluster(self):
        self.clusters
```

Con estas clases a la hora de buscar un objeto la complejidad se baja a $O(\log n)$, en el caso de las clases nodo y en el caso de clusterinfo es para solucionar problemas de caché del servidor y de los navegadores, además de devolver información importante del análisis con Kmeans.

9.0.9. Variables de análisis

- **numeroOrganizaciones** Esta variable es un diccionario que tiene como nodo1 cada vez que un tweet se analiza. Si Spacy encuentra una organización, se crea una variable de tipo nodo1 y en el caso de que en el diccionario no exista esa palabra se introduce, si existe, se busca en el diccionario y en nodo1 hay una función que es sumar para aumentar en uno las veces que ha aparecido esta palabra en los tweets.

```
if str(token2.label_) == "ORG":
    if numeroOrganizaciones.get(token2.text) is None:
        n = nodo1(token2.text)
        numeroOrganizaciones[n.palabra] = n
    else:
        numeroOrganizaciones[token2.text].sumar()
```

La variable token es un iterador de texto2, esta variable es el análisis creado con Spacy, iremos recorriendo y dentro de la variable token tiene dos argumentos, label_ contiene las siglas ORG para las organizaciones, LOC para las localidades, PER para las personas famosas, solo recorreremos la variable y con un if vamos comprobando y lo guardamos en el diccionario.

- **numeroLocalidades** Igual para este caso en el que vamos recorriendo el análisis de Spacy y vamos detectando las palabras LOC, seguido al detectarlo se introduce al diccionario.

```
if str(token2.label_) == "LOC":
    if numeroLocalidades.get(token2.text) is None:
        n = nodo1(token2.text)
        numeroLocalidades[n.palabra] = n
    else:
        numeroLocalidades[token2.text].sumar()
```


- **numeroPersonas** La forma de encontrar las personas se ha utilizando la biblioteca Spacy y buscando en los resultados en las que su argumento sea 'PER' y se guardan en este diccionario.

```
if str(token2.label_) == "PER":
    if numeroPersonas.get(token2.text) is None:
        n = nodo1(token2.text)
        numeroPersonas[n.palabra] = n
    else:
        numeroPersonas[token2.text].sumar()
```

- **numeroHastag** Para el caso de los hashtag antes de introducir el texto en Spacy se recorre y se va detectando en cada palabra si contiene "#" para guardar en este diccionario.

```
if v.find('#') != -1:
    val = v.find("#")
    if val != -1:
        v = v[val:]
    if numeroHastag.get(v) is None:
        n = nodo1(v)
        numeroHastag[n.palabra] = n
    else:
        numeroHastag[v].sumar()
```

- **numeroUsuarios** Para el caso de encontrar usuarios dentro de los textos, se buscarán @ y se añadirán en el diccionario.

```
if v.find('@') != -1:
    val = v.find("@")
    if val != -1:
        v = v[val:]
    if numeroUsuarios.get(v) is None:
        n = nodo1(v)
        numeroUsuarios[n.palabra] = n
    else:
        numeroUsuarios[v].sumar()
```

- TotalWikipedia Cada palabra se pasa por el conjunto de Wikipedia, y se le da a cada tweet una puntuación, se crea una variable de tipo nodo4 y se guarda en el diccionario.

```
n4Wikipedia = nodo4(line['id_tweet'],line['full_text'],
puntuacionWikipedia,
line['URLTweet'],listaWikipedia,line['NombreUsuario'])
TotalWikipedia.append(n4Wikipedia)
```

- TotalFichero Exactamente igual que en el apartado anterior, se compara cada palabra de cada tweet, pero esta vez con el conjunto de palabras introducidas por el usuario. Se le da la puntuación a las que contengan las palabras que hay en el conjunto, se crea una variable de tipo nodo4 y se guardan en el diccionario TotalFichero.

```
n4Fichero = nodo4(line['id_tweet'],line['full_text'],puntuacionFichero
,line['URLTweet'],listaFichero,line['NombreUsuario'])

TotalFichero.append(n4Fichero)
```

9.0.10. Funciones creadas

- CreacionDiccionario(partes)

Después de realizar un tratamiento sobre el tema búsqueda, lo ponemos en minúscula y quitamos las palabras basura que puedan contener. Luego lo introducimos en creacionDiccionario, se realizará una consulta a Wikipedia sobre el termino, realizando una captura de la consulta y después nos quedamos con los sustantivos, metemos todos los sustantivos, pronombres, organizaciones en un conjunto con el que se comprobaran las palabra que tiene cada tweet.

La función summary de Wikipedia tenemos que introducir en ella la consulta y el tamaño del texto, las páginas de Wikipedia contienen mucha información pero la que nos interesa es la información relacionada con el significado del texto. En esos primeros párrafos es donde están el mayor número de palabras relacionadas con la búsqueda. Después de esto mandamos el texto que nos devuelve la Api a la función clasificador de entidades.

La función CreacionDiccionario tiene las siguientes observaciones, se ha encontrado muchos problemas a la hora de la implementación de esta función debido a los tipos de errores que puede tener la Api de Wikipedia, van a explicarse cada uno y el tipo de excepción que se ha utilizado para subsanarlos.

- `wikipedia.exceptions.DisambiguationError`: Este error ocurre por no devolver una lista de artículos relacionados, no nos sirve para este estudio pero con algunos términos ha fallado y desde que se puso se subsanaron muchos errores.
- `wikipedia.exceptions.PageError` En el caso de que exista el artículo pero las páginas estén en mantenimiento o vacías.
- `requests.exceptions.ConnectionError` En el caso de que se haya utilizado el número máximo de conexiones en la Api.
- El último except es en el caso de que se ponga un valor que no exista en la Wikipedia.

En todos los casos en los que las Api de Wikipedia no devuelve el contenido de la consulta, se crea un conjunto de valores vacíos. **Valores de entrada** el texto que se introduce de búsqueda general. **Valores de salida** un conjunto con los sustantivos para realizar la búsqueda.

```
def CreacionDiccionario(valor):
    try:
        try:
            wikipedia.set_lang("es")
            final = set()
            try:
                try:

                    tex = wikipedia.summary(valor, sentences=5)
                    print(tex)
                    solucion = ClasificarEntidades(tex)

                    return solucion

            except wikipedia.exceptions.DisambiguationError:
                print("error wikipedia")
                solucion = set()
                solucion.add("No se ha encontrado nada en wikipedia")
                return solucion

        except wikipedia.exceptions.PageError as e:
```

```

        print("error no existe esa pagina en wikipedia")
        solucion = set()
        solucion.add("No se ha encontrado nada en wikipedia")
        return solucion
    except requests.exceptions.ConnectionError:
        requests.status_code = "Maximo de intentos cumplidos."
        solucion = set()
        solucion.add("No se ha encontrado nada en wikipedia")
        return solucion

    solucion = set()
    solucion.add("No se ha encontrado nada en wikipedia")
    return solucion
except:
    solucion = set()
    solucion.add("No se ha encontrado nada en wikipedia")
    return solucion

```

■ ClasificarEntidades(tex)

I Los valores de entrada son el texto que nos ha devuelto la Api de Wikipedia y realizamos un análisis con Spacy y nos quedamos con las organizaciones, las personas, los pronombres y los sustantivos. En el caso de los pronombres y los sustantivos los lematizamos para que así quitar las conjugaciones y quedarnos con la raíz. Todo esto lo introducimos en un conjunto que es el **valor del salida**.

\newpage

```

def ClasificarEntidades(valor):
    nlp = spacy.load("es_core_news_sm")

    texto = nlp(valor)
    diccionario = set()

    for token in texto:

```

```
if str(token.pos_) == "ORG":
    temp = str(token.text).lower()
    diccionario.add(temp)
if str(token.pos_) == "PERSON":

    temp = str(token.text).lower()
    diccionario.add(temp)
if str(token.pos_) == "PROPN":

    temp = str(token.text).lower()
    temp2 = lematizar(temp)
    diccionario.add(temp2)

if str(token.pos_) == "NOUN":

    temp = str(token.text).lower()
    temp2 = lematizar(temp)
    diccionario.add(temp2)

return diccionario
```

- `Análisis(diccionarioWikipedia, diccionarioFichero, data)`

Los valores de entrada son los dos conjuntos creados tanto el de Wikipedia como el de los valores que introduce y los datos capturados de Twitter.

Recorremos los tweets y cada palabra la lematizamos, vemos si esta contenida en los dos conjuntos el de las palabras de Wikipedia y el conjunto de palabras que ha introducido el usuario al principio de la captura, después de esto con cada hallazgo de alguna palabra se suma al tweet un punto para luego después realizar la gráfica correspondiente.

La función `NOSTopWords` es para verificar si es una palabra vacía, devuelve verdadero o falso dependiendo si está en conjunto de valores innecesarios.

```
def NOSTopWords(valor):
```

```
if valor not in stop_words:
    return True
else:
    return False
```

Antes de compararlas con los conjuntos de Wikipedia y con los valores introducidos por el usuario hay que lematizar cada palabra antes.

```
def lematizar(valor):
    return (spanish_stemmer.stem(valor))

for i in range(tamano):
    if NOStopWords(vector[i]):
        transformacionLematizada = lematizar(vector[i])
        if str(transformacionLematizada).lower() in diccionarioWikipedia:
            puntuacionWikipedia += 1

            listaWikipedia = listaWikipedia + " " + vector[i]

        if str(transformacionLematizada).lower() in diccionarioFichero:
            puntuacionFichero += 1
            listaFichero = listaFichero + " " + vector[i]

    n4Wikipedia = nodo4(line['id_tweet'],line['full_text'],
        ,puntuacionWikipedia,line['URLTweet'],
        ,listaWikipedia,line['NombreUsuario'])
    TotalWikipedia.append(n4Wikipedia)
    n4Fichero = nodo4(line['id_tweet'],line['full_text'],
        ,puntuacionFichero,line['URLTweet'],
        ,listaFichero,line['NombreUsuario'])

    TotalFichero.append(n4Fichero)

    texto = texto+" "+98765432+" "+str(line['full_text'])
```

Observaciones: Spacy tiene un límite a la hora de realizar análisis para encontrar entidades dentro del texto

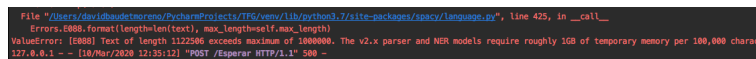


Figura 9.33: Error límite de análisis por Spacy

El máximo de caracteres que es capaz de gestionar Spacy para un análisis es de 1.000.000, en caso de ser mayor tenemos que fraccionar el conjunto de datos.

Se recorren los tweets hasta que se terminan o cuando el tamaño del número de caracteres es mayor a 900.000, se lanza la aplicación que realiza el análisis de tweets con Spacy, en el caso de sobrepasarlo se pone una variable centinela a 1 para tener en cuenta el resto de caracteres menores.

```
if len(texto) > 900000:
    analizarTexto(texto)
    centinela = 1
    texto=""

if centinela == 1 and len(texto) != 0:
    analizarTexto(texto)

elif centinela == 0 and len(texto) != 0:
    analizarTexto(texto)
```

Valores de salida ninguno se llama a la función guardarDatos() que genera los ficheros con los análisis para después representarlos en gráficas de Bokeh.

- **Clustering(archivo)** Los **valores de entrada** son el archivo de captura de datos, para la realización del estudio de agrupamiento se han utilizado dos librerías Sklearn que contiene el algoritmo Kmeans. Kneed es para utilizar el método del codo y generar un gráfico con el mejor número de cluster para aplicarlo en Kmeans. El tamaño de la captura debe superar los 300 tweets para realizar el análisis con las herramientas kneed, en el

caso de no tener este número de tweets se genera unas gráficas vacías en las que el título es 'error mínimo de tweets'. Esto es para que el usuario tenga visibilidad de la razón de que la gráfica no se genere.

Si cumple el tamaño de la captura, se realiza un tratamiento de la información quitando las palabras vacías y pasando todas las palabras a minúscula.

Después se construyen dos listas una con los tweets de entrenamiento de Kmeans y otros con los de prueba. Para el aprendizaje se toma un 80 % de los tweets y para la prueba se cogen el 20 % restante.

```
aprendizaje = data.__len__() * 0.8
for c in completo:
    if bandera < aprendizaje:
        frases.append(c)

    else:
        prueba.append(c)
bandera += 1
```

Después de esto vectorizamos las palabras de aprendizaje, realizamos un bucle de 1 a 10 cluster para generar la gráfica en la que se ve la función y en ella detectamos el punto de codo para el elegir el número de cluster óptimo.

En el caso de no encontrar punto de codo se coge por defecto un total de cinco cluster por ser el valor medio de de 1 a 10 cluster.

Se añade los texto al eje x "número de cluster" y en el eje y "Suma de distancias al cuadrado".

Ya tenemos nuestro valor óptimo para Kmeans, realizamos el aprendizaje en el algoritmo y seguidamente aplicamos el conjunto de datos de prueba, para terminar realizamos una prueba con el conjunto de datos completo, para así tener constancia en cual cluster esta cada tweet.

Además realizamos una lista de las palabras más representativas de cada cluster, estos valores se introducen en una lista que se guarda en la variable de clase clusterinfo.

Lo guardamos en el fichero Clustering.csv, por último se observó un problema a la hora de generar el gráfico del método del codo, en el cual se repetía el gráfico aunque la consulta del usuario fuera distinta. Esto es debido a que los navegadores tienen caché,

guardan la foto para no tener que volver a descargarla de un servidor. La solución que se ha tomado ha sido no guardar la foto como un archivo, sino guardarla en una variable de clase clusterinfo que se devuelve con la función clustering. De esta forma, se evita el problema de la caché en la gráfica, más adelante se explicará en el apartado de la página web como se ha subsanado el resto de problemas que ha dado la caché.

Para el caso de que no haya un número de tweets suficientes para realizar el agrupamiento, se crea una variable clusterinfo que está vacía y con el gráfico antes mencionado en que se pone error en tamaño de la captura.

```
def Clustering(archivo):
    data = pd.read_csv(archivo, sep='\t', engine="python" , quotechar='\"'
    , error_bad_lines=False)
    frases = []
    prueba = []
    cl=""
    cl = list()

    completo = []
    URLs = []
    id = []
    final = []

    numero = data.__len__()

    if numero > 300:

        aprendizaje = data.__len__() * 0.8

        for i , d in data.iterrows():

            linea = str(d['full_text']).lower().split()
            lineaFiltrada = []
            for palabra in linea:
                p = palabra.lower()
```

```
        if p not in stopwords.words('spanish'):
            lineaFiltrada.append(p)
        lineaLimpia = ' '.join(lineaFiltrada)

    completo.append(lineaLimpia.replace('\n','').replace("\r",""))
    URLs.append(d['URLTweet'])
    id.append(d['id_tweet'])

bandera = 0
for c in completo:
    if bandera < aprendizaje:
        frases.append(c)

    else:
        prueba.append(c)
    bandera += 1

#-----

cv = CountVectorizer()

vectorizer = TfidfVectorizer(stopwords.words('spanish'))
X = vectorizer.fit_transform(frases)
data2 = cv.fit_transform(frases)

distortions = []
K = range(1,10)
for k in K:

    model = KMeans(n_clusters=k)
    model.fit(data2)
    distortions.append(model.inertia_)
```

```

x = range(1, len(distortions) + 1)

kn = KneeLocator(x, distortions,
curve='convex', direction='decreasing')

if kn.knee is None:
    true_k = 5
else:
    true_k = kn.knee

plt.xlabel('numero de clusters k')
plt.ylabel('Suma de distancias al cuadrado')
plt.plot(x, distortions, 'bx-')
plt.vlines(kn.knee, plt.ylim()[0],
plt.ylim()[1], linestyle='dashed')

#-----

true_k = kn.knee

model = KMeans(n_clusters=true_k,
init='k-means++', max_iter=100, n_init=1)
model.fit(X)

order_centroids = model.cluster_centers_.argsort()[:, :-1]
terms = vectorizer.get_feature_names()

for i in range(true_k):
    v=""
    v = "cluster "+str(i)

    for ind in order_centroids[i, :10]:

```

```

        v = v + ","+str(terms[ind])

    cl.append(v)

for p in prueba:
    X = vectorizer.transform([p])
    predicted = model.predict(X)

a = 0
datos = open('FicherosSalida/Clustering.csv', 'w')
datos.write("id"+"\\t"+"texto"+"\\t"+"cluster"+"\\t"
+"URL"+"\\n')
while a < len(completo):
    X = vectorizer.transform([completo[a]])
    predicted = model.predict(X)

    cluster = str(predicted).replace('[', "").replace(']', "")
    datos.write(str(id[a])+'\\t'+str(completo[a])+
'\\t'+str(cluster)+'\\t'+str(URLs[a])+'\\n')
    a += 1
datos.close()

salida = info(plt,len(completo),len(frases),len(prueba),cl)

return salida

else:

a = 0
datos = open('FicherosSalida/Clustering.csv', 'w')
datos.write("id" + '\\t' + "texto" + '\\t'
+ "cluster" + '\\t' + "URL" + '\\n')
datos.close()
plt.xlabel('error numero de minimo de datos')
plt.ylabel('error numero de minimo de datos')

```

```

cl = list()

salida = info(plt, 0, 0, 0,cl)

return salida

```

9.0.11. Estructura de el análisis de datos

La estructura conforme se han llamado a estas funciones antes nombradas y también las de bokeh es la siguiente:

```

diccionarioWikipedia = CreacionDiccionario(partes)
Analisis(diccionarioWikipedia, diccionarioFichero, data)
archivo = 'FicherosSalida/Fichero.csv'
info = Clustering(archivo)

data3 = pd.read_csv('FicherosSalida/Clustering.csv'
, sep='\t', engine="python")
cluster = data3["cluster"].tolist()
#####
Estado = "Creación de los graficos"
figura2 = bokeh2(archivo)
figura3 = bokeh3(archivo)
figura4 = bokeh4(archivo)
figura7 = bokeh5('FicherosSalida/PuntuacionWikipedia.csv')
figura8 = bokeh6('FicherosSalida/PuntuacionFichero.csv')
figura5 = bokeh12('FicherosSalida/Hastag.csv')
figura6 = bokeh9('FicherosSalida/Usuarios.csv')
figura13 = bokeh13('FicherosSalida/Localidades.csv')
figura10 = bokeh11('FicherosSalida/Clustering.csv')
figura11 = bokeh14('FicherosSalida/Organizaciones.csv')

```

Después de esto se cargarán todas las figuras en las respectivas páginas web que más adelante en el apartado de web se explicará.

9.0.12. Experimentos

La cantidad de experimentos que se ha realizado durante la ejecución del proyecto han sido ———— estos experimentos durante la pruebas de los algoritmos por separado, fueron sobre temas de catástrofes, inundaciones y de carácter social como podía ser los disturbios en Barcelona. Se buscaban focos de tweet de mucha cantidad para probar la robustez de las funciones.

Las carpetas están divididas en dos categorías, las capturas antes de tener el análisis de los datos con la biblioteca Bokeh que son archivos html y las otras carpetas son archivos csv o json, con las capturas de Twitter.

Dentro de la carpeta de experimentoshtml, hay dos versiones, esto es por cambios que se realizaron en las páginas web a la hora de colores en los punto de las gráficas. Se cambiaron a azul en una tonalidad oscura para su buena distinción.

Con el TFG se va adjuntar unas carpetas con todos los experimentos realizados.

Capítulo 10

Desarrollo de un portal web para pruebas

Para el desarrollo del portal web se ha utilizado un microframework de Python llamado Flask, debido que la página web es una página de pruebas se ha realizado una página sencilla para lanzar pruebas de funciones.

10.0.1. Flask

La instalación se realiza de manera muy cómoda utilizando el comando pip3:

```
pip3 install Flask
```

Flask es una microframework que utiliza el lenguaje Python, el motivo por el que se ha utilizado esta librería es por que así puedo lanzar las funciones de Python desde el mismo servidor de Flask. Hay otras alternativas como Django pero no se han contemplado por la dificultad de aprendizaje respecto a Flask.

Al tener en el servidor alojado en una plataforma con el lenguaje Python es sencillo lanzar las aplicaciones a un determinado tiempo. Aunque unos de los problemas que se ha tenido a sido el visualizar estados y datos de las funciones en la vista del servidor de Python, se ha introducido cambios en la interface web mientras que se está realizando la captura para así aumentar el feedback con el usuario, para que no se lance una consulta y no se vea la ejecución de las funciones.

10.0.2. Características más importantes de Flask

Flask es tiene un patrón de programación MVC modelo vista controlador, estas son algunas de las características más importantes de Flask:

- **Jinja template**

Esta característica nos brinda la posibilidad de utilizar plantillas para generar las vistas, con ello minimizamos el número de páginas que se crean a unas pocas. Además este sistema de Jinja tiene propiedades de herencia entre plantillas útil para el heredar menús entre las plantillas. En este proyecto se ha utilizado herencia entre plantillas para el caso de la representación de la página principal, se le ha integrado un menú de navegación con una introducción y la opción de captura de datos.

- **WSGI**

Interfaz de puerta de enlace para reenviar peticiones web al servidor, es básico tener esta característica debido a que sino, el usuario cuando pinchara en un enlace el servidor no redireccionaria ninguna vista.

- **Trae un depurador propio para el servidor todo esto junto a que la aplicación con la que se ha desarrollado el proyecto Pycharm tiene la opción de un proyecto en Flask, esto ayuda al programador con más herramientas de depuración.**

- **Una gran comunidad**

Ha habido inconvenientes con el servidor en varios aspectos, debido a la caché que genera el propio servidor y la que genera los navegadores al realizar las peticiones a este. Con la ayuda de la documentación y de la comunidad se ha subsanado el problema de la caché tanto en navegadores como en el servidor.

- **Extensiones de flask**

Un gran conjunto de extensiones para el servidor que ayudan a integrar mejor nuevas características al proyecto, para este caso de estudio se ha integrado una de estas extensiones que ayuda al manejo del caché del servidor. Esto se analizará más adelante junto a otras directrices que se han utilizado para subsanar este problema.

- **Sesiones**

Flask es compatible con el uso de sesiones, en el caso de este proyecto no se han utilizado las sesiones porque el objetivo es realizar una web de pruebas para la visualización de los datos capturados y posteriormente analizados.

10.0.3. Lanzar flask

Para lanzar el servidor primero debemos crear una instancia del servidor y aplicar un nombre.

```
app = Flask(__name__)
```

En el caso de este proyecto, se añadió la inclusión de la carpeta que contiene los archivos.html. Estas son las plantillas que se generan para la vista del servidor.

```
app = Flask(__name__, template_folder="templates")
```

Seguido de esto hay que introducir una dirección ip para nuestro servidor para poder acceder a él y un puerto de acceso.

```
app.run(host=os.getenv('IP', '0.0.0.0'),port=int(os.getenv('PORT', 4554)))
```

Una observación sobre las ip es que hay que tener en cuenta que si se quieren implementar varios proyectos a la vez, estos deben tener direcciones ip distintas o puertos distintos. En el caso de tener la misma dirección ip y mismo puerto, la aplicación da una excepción.

Otra problema que puede surgir es el tener conexión a internet, si algún componente necesita internet, dará un error en el caso de no capturar la excepción. Si el componente es de un aspecto visual como un css o bootstrap este no se cargará pero no dará error.

En el caso de este proyecto los menús están generados con componentes que se descargan de la página de W3schools, además de que la Api de Twitter necesita internet para funcionar. Para solventar este problema se realiza un ping a Twitter para comprobar la conexión.

```
hostname = "www.twitter.com"
response = os.system("ping -c 1 " + hostname + " &> /dev/null")
```

En el caso de que response contenga 0 se lanza el servidor, en cualquier otro caso se muestra un mensaje de red caída o Twitter caído.

El peor problema con el que se ha tenido que lidiar a la hora de realizar la página web, sin duda ha sido la caché del usuario y del navegador. Se han tomado las siguientes directrices para evitar dicho problema:

- Flask_caching

Instalar en el servidor este modulo de flask que no viene el propio microframework para su instalación. Hay utilizar pip3 en el propio entorno virtual del proyecto.

Una vez instalado esta extensión de Flask tiene varios modos de actuación sobre el servidor:

- Null:

No guarda ninguna caché sobre el servidor que se está utilizando, esta opción es la que se ha utilizado en este proyecto. El motivo de esta configuración es que se han lanzado multitud de experimentos con el mismo tema de búsqueda: catástrofes, inundaciones, coronavirus y COVID-19. Debido a esto, si no se pone esta configuración el primer experimento con la palabra COVID dará unos resultados correctos, en el segundo que se lance con el mismo tema de búsqueda muchos de las partes del análisis será igual que en el anterior análisis, el más notable entre ellos son las gráficas para la técnica del codo en el apartado de agrupamiento.

- SimpleCache:

Es el nivel de caché por defecto en esta extensión, en el caso de activar este modo algunas partes del análisis se repiten.

- FileSystemCache:

Es para aplicar una caché sobre los archivos que se genera en la página web, no se ha utilizado.

- RedisCache:

Este nivel de caché es para la hora de darle segundos de cache a la conexión de los usuarios.

- El resto de cache son aplicados a la memoria del servidor.

El objetivo del servidor web que se ha utilizado es para poder realizar pruebas de las funciones implementadas en el caso de realizar una aplicación final, tendrían que tenerse en cuenta estos niveles de caché.

Al declarar esta app en el main hay que introducirle el tipo de caché que vamos a utilizar.

```
app = Flask(__name__, template_folder="templates")
app.config['CACHE_TYPE'] = 'null'
app.cache = Cache(app)
```

- Nombres de los archivos:

Para evitar que el usuario cuando se realice el final del análisis descargue este igual que los anteriores por el problema de la caché, se guarda cada archivo que se crea en el servidor con el nombre del tema de búsqueda para así evitar que la caché del navegador haga que no se descargue el archivo correcto. El tema de la caché del navegador de cada usuario es un problema a la hora de aplicaciones de este tipo, ya que el servidor no tiene permisos para borrar la caché de los navegadores de cada usuario, para solucionar este problema hay varias opciones:

- Avisar al usuario para que realice un borrado de la caché del navegador.
 - Utilizar una Api como puede ser selenium para borrar la caché del navegador desde el servidor, lo que conllevaría la detección del tipo de navegador del usuario Firefox, Opera, Safari, Cortana. En cada uno tiene una implementación en esta Api para realizar un borrado de la caché.
- Guardar en memoria los plot:
- A diferencia de los gráficos generados por Bokeh que son todos distintos, los archivos plot generados con la librería Matplotlib. El servidor no es capaz de distinguir un archivo creado de uno antiguo, incluso cambiándole el nombre del archivo sigue sin detectarlos, para solucionar este problema se implementó la clase clusterinfo, esta contiene el gráfico en unos de sus atributos. Esta es la salida de la función del agrupamiento, el atributo plt es el gráfico que contiene la función del codo antes descrita.

```
salida = info(plt,len(completo),len(frases),len(prueba),cl)

return salida
```

Hasta aquí tenemos ya el plot en la vista, ahora tenemos que convertir la variable plot en una imagen para integrarla en la página web.

El problema es que no podemos guardar en un fichero PNG la imagen, porque el servidor no utilizará la que tiene en caché. Para solventar este problema no la guardamos en un fichero sino en una variable que se la pasamos a la función render_template junto con el template con los resultados.

Para convertir el plot a imagen sin guardarla, tenemos que pasar los bits del plot a PNG, con la función BytesIO y seek. Al utilizar BytesIO se abre un contenedor para guardar una matriz de bits, se introduce después el plot pasándolo al formato PNG y después se cierra la variable con seek. Tenemos los bits de la imagen en una variable pero tenemos que guardarla en un formato que el html lo pueda leer, para esto lo decodificamos en ascii.

```
figfile = io.BytesIO()
info.plot.savefig(figfile, format='png')
figfile.seek(0)
figdata_png = base64.b64encode(figfile.getvalue()).decode('ascii')
```

```
im2 = figdata_png
```

En im2 tenemos ya el gráfico listo para usarse en una página web y con esto solventamos el problema de la caché.

10.0.4. Vistas creadas y funciones creadas

Cada vez que el usuario navega por la aplicación web o lanza una captura, el servidor Flask lanza una de las plantillas de la carpeta templates, por consiguiente al lanzar la aplicación hay que cargar este directorio:

```
app = Flask(__name__, template_folder="templates")
```

Para lanzar y redireccionar una plantilla se utiliza la función `render_template`, esta función la contiene el propio microframework flask, esta función se le pueden pasar como parámetros además del archivo.html, el título.

Cada función o cada vista que se quiera lanzar desde una plantilla tiene que tener el siguiente formato:

Primero `@app.route()` esto es el redireccionador de direcciones que tiene Flask. Entre los paréntesis estará la ruta de la aplicación o de la vista nueva a cargar, luego además tiene dos variantes dependiendo de si los valores son de entrada o de salida.

- `@app.route('/procesar', methods=['POST'])`

En este ejemplo se va a lanzar la vista procesar y al tener el método POST nos indica que esta redireccionando una plantilla de la carpeta, que manda información además de la plantilla en este caso los valores introducidos por el usuario.

- `@app.route('/Estado')`

Para las funciones o vistas que no mandan valores no hace falta incluir POST, en este caso devuelve un valor a la misma vista que llama a la función.

- `@app.route('/')`

Al acceder a la página web el usuario lanza esta vista, el servidor redirecciona el template index.html, el cual contiene la página principal de la aplicación.

```
@app.route('/')
def home():
    return render_template('/index.html',
                           title="Análisis de eventos en twitter")
```

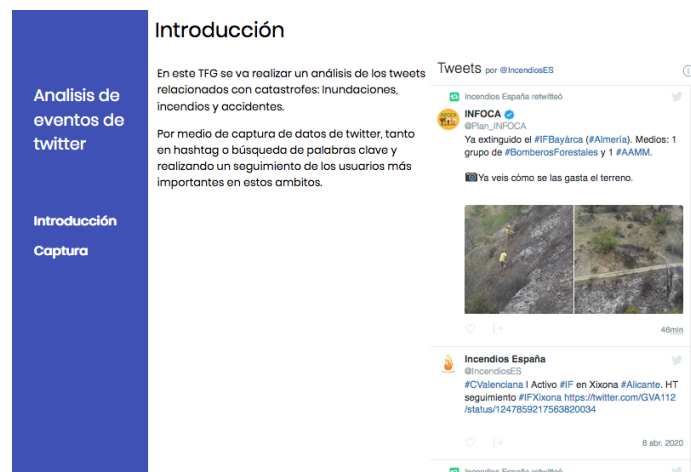


Figura 10.1: Pagina principal

- `@app.route('/captura')`
 Cuando el usuario pincha en el menú en el apartado Captura, esta vista redirrecciona a la página de captura de datos de entrada.

- `@app.route('/contador_tweets')`
 Para aumentar el feedback con el usuario se ha introducido esta función, un script escrito en ajax en la plantilla Espera, llama a esta función devuelve a la plantilla el número de tweets que ya hay descargados.
 Para conseguir el número de tweets descargados en ese momento, se ha creado una función de apoyo dentro de las funciones de captura, `get_contador` retorna el número de tweets descargados hasta ahora.

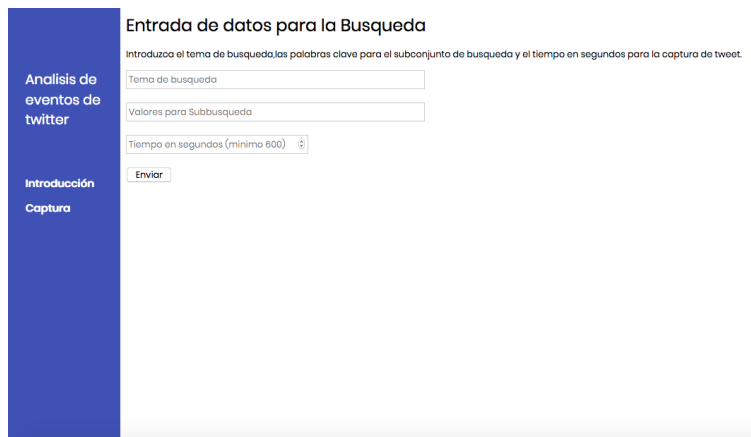


Figura 10.2: vista captura

```
@app.route("/contador_tweets")
def live_game():
    a= get_contador()
    return "<p> " + str(a) + " </p>"
```

```
def get_contador():
    global contador
    return contador
```

Como observación para actualizar el valor en la página web hay que convertir el número de texto y después pasarlo como un párrafo html.

- `@app.route(/tweets")` Otra medida para darle más feedback, es mostrar los tweets ya descargados en un `textArea` introducido en `Espera.html`, un script en ajax llama a esta función para mostrar los tweets descargados.

Para conseguir los tweets que hay guardados en este momento, se implementó una función de apoyo dentro de las funciones de captura de tweet, `get_tweets` devuelve el texto de los tweets y el usuario que los escribió.

```
@app.route("/tweets")
def live_game222():
    salida = get_tweets()
    return str(salida)
```

```
def get_tweets():  
    global tweets  
    return tweets
```

- `@app.route('/Estado')` Esta función es para pasar el estado de la ejecución del programa, durante la ejecución de la captura y del análisis se modifica una variable string que contiene el estado, esta función devuelve su contenido a plantilla `Espera.html`, esto es para ayudar al usuario a saber el estado de la ejecución. Sus estados son los siguientes:
 - Empezamos la captura de tweet
 - Capturando tweets
 - Empezando el Análisis
 - Creando diccionarios de Wikipedia y de valores de entrada
 - Creando gráficos
- `@app.route('/procesar', methods=['POST'])` Esta función es para realizar un procesado de los datos de entrada que realiza el usuario y los manda junto con la plantilla `Espera.html`.

```
@app.route('/procesar', methods=['POST'])  
def procesar():  
  
    global tema, valores, tiempo, tiempo2, abortar  
    tema = request.form.get("tema")  
    valores = request.form.get("valores")  
  
    tiempo = request.form.get("tiempo")  
    tiempo2 = request.form.get("tiempo")  
    abortar = 0  
    return render_template("/Espera.html",  
        tema = tema, valores = valores,  
        tiempo = str(tiempo2))
```

- `@app.route('/download2')` Esta función se activa cuando se termina el análisis de los datos y el usuario quiere descargar los datos obtenidos.

```
@app.route('/download2')
def download_file():
    import matplotlib
    matplotlib.use('Agg')
    global paginaDescarga,tema
    path = tema+".html"
    return send_file(path, as_attachment=True)
```

Para no tener problemas con la caché del servidor, el archivo que se devuelve será el que el usuario luego descargará, teniendo el nombre del tema de búsqueda de tweets. Durante las primeras ejecuciones de código que se realizaron, el programa se cerraba, tras cargar la plantilla resultados2.html. El problema es de la función `send_file` que es propia del microframework, para subsanar este problema se tiene que añadir la línea de código siguiente:

```
import matplotlib
matplotlib.use('Agg')
```

Es problema del framework, esta solución fue dada por la comunidad de Flask.

- `@app.route('/abortar', methods=['POST'])`
En la plantilla `Espera.html` tiene un botón para abortar la ejecución de descarga de tweet, este botón se introdujo para las pruebas de desarrollo que se realizaron.

Para parar la descarga de tweets se utiliza una variable llamada `abortar`, esta variable está iniciada con el valor 0. En caso de que el usuario pinche en el botón se cambia a 1 y se realiza una parada de las funciones de descarga de tweet.

```
@app.route('/abortar', methods=['POST'])
def abortar():
    global abortar
```



```
abortar = 1
return render_template('/index.html', title="Introducción")
```

- @app.route('/Esperar', methods=['POST'])

Esta vista es la más compleja, debido a que tiene como propósito la captura y el análisis de los tweets, se va a desgantar la vista por partes para así su mejor comprensión.

Al lanzar la función provoca una limpieza de los contenedores, esto realiza una limpieza de las variables contenedoras de los tweets.

```
limpiar_tweets()
limpiar_contador()
limpiar_contenedores()
```

- Limpiar_tweets:

Limpiar la variable tweets de la funciones de captura de tweet, esto es para que al inicio de una captura nueva no aparezcan por pantalla ningún tweet de un experimento anterior.

- Limpiar_contador:

Resetea el valor de los tweets descargados.

- Limpiar_contenedores():

Borra los tweets que almacena la lista y borra los items del conjunto de id de tweets.

Introducimos en la variable que lleva el estado de la ejecución que ha empezado la captura de tweets.

Guardamos en la variable usuarios, la captura de usuarios relacionados con el tema de búsqueda.

```
usuarios = CapturarUsuarios(tema)
```

Después realizamos una captura de tweets con la función Busqueda, Busqueda2, analisisUsuarios y AlertaHashtag. En el caso de que se encuentre un hashtag relacionado con el tema de búsqueda, se lanza la función búsqueda con este valor.

Después de realizar llamadas iniciales, empieza el bucle que tendrá la duración que haya sido introducida por el usuario.

```
while (time.time() - tiempoejecucion) < limit and abortar != 1:
    Estado = "Capturando tweets"
    reloj = clock()
    if reloj - tres > 3 * 60 and abortar != 1:
        CapturaTiempoReal(tema)
        if abortar != 1:
            Busqueda(tema)
            tres = reloj
    if reloj - cinco > 5 * 60 and abortar != 1:
        Busqueda2(tema)
        cinco = reloj
    if reloj - nueve > 9 * 60 and abortar != 1:
        if len(usuario) != 0 or usuario != None:
            AnalisisUsuario(usuario, tema)
        valor = AlertaPorHashtag(tema)
        if valor != "":
            print("valor encontrado en Hashtag",valor)
            Busqueda(valor)
        nueve = reloj
```

Las variables tres, cinco y nueve son contadores para cada bloque temporal para la hora de lanzar las funciones. Cada 3, 5 y 9 minutos, limit es el tiempo máximo que se estará ejecutando el bucle. Tiempoejecucion es el tiempo de inicio del bucle, que se toma de la librería time, reloj es una variable que se actualiza con la hora del sistema, esta actualiza las variables: tres, cinco y nueve para que se ejecuten periódicamente las funciones que contienen cada bloque.

Una vez se ha terminado el proceso de captura de tweets, se lanza la aplicación GuardarYSalir(), que genera un Fichero.csv y Foco.csv. Estos contienen los datos capturados y el primer tweet escrito sobre el tema introducido, para proseguir con su análisis.

Cambiamos el estado para que en la plantilla que ve el usuario observe el avance de la ejecución, una vez que se ha actualizado el estado de ejecución, se va a crear los conjuntos de palabras con los que se va a cotejar si las palabras los contienen o no.

Para el tratamiento se transforma el texto en minúscula y quitamos las palabras vacías que pueda contener la entrada de búsqueda.

```
Entrada = tema
Entrada = Entrada.lower()
vectorEntrada = Entrada.split(" ")
transformado = ""
partes = list()
for vE in vectorEntrada:
    if NOStopWords(vE):
        partes.append(vE)
Estado = "Creando diccionarios de wikipedia y de valores de entrada"
diccionarioWikipedia = CreacionDiccionario(partes)
```

En `diccionarioWikipedia` es un conjunto con los valores encontrados en la web de Wikipedia y posteriormente se han sustraído los sustantivos, organizaciones y pronombres.

Con los valores escritos por el usuario para la subbúsqueda dentro de los tweets hay que realizar también un tratamiento para crear el conjunto con el que cotejar los tweets.

Cada palabra del subconjunto se divide por comas, se convierte a texto en el caso de que haya algún valor numérico, después se lematiza para quedarnos solo con las raíces.

```
valores = str(valores)
lista = valores.lower().split(',')
diccionarioFichero = set()
for l in lista:
    diccionarioFichero.add(lematizar(l))
```

Al acabar la generación de los dos conjuntos tanto el de Wikipedia como el de los valores, se envían junto con los datos capturados.

```
Analisis(diccionarioWikipedia, diccionarioFichero, data)
```

La siguiente etapa del análisis es cambiar la variable Estado para actualizar el momento del análisis, a continuación se aplica un algoritmo de agrupamiento supervisado.

```
Estado = "Aplicando algoritmo de agrupamiento"  
info = Clustering(archivo)
```

Tras la ejecución del algoritmo de agrupamiento no supervisado se carga la imagen en una variable como antes se ha explicado.

```
if info.plot is not None:  
    figfile = io.BytesIO()  
    info.plot.savefig(figfile, format='png')  
    figfile.seek(0)  
    figdata_png = base64.b64encode(figfile.getvalue()).decode('ascii')  
  
    im2 = figdata_png
```

Para terminar se empieza con la creación de cada gráfico con Bokeh, que se mostrará en los resultados.

```
Estado = "Creando gráficos"  
figura2 = bokeh2(archivo)  
figura3 = bokeh3(archivo)  
figura4 = bokeh4(archivo)  
figura7 =bokeh5('FicherosSalida/PuntuacionWikipedia.csv')  
figura8 =bokeh6('FicherosSalida/PuntuacionFichero.csv')  
figura5 = bokeh12('FicherosSalida/Hastag.csv')  
figura6 = bokeh9('FicherosSalida/Usuarios.csv')  
figura13 = bokeh13('FicherosSalida/Localidades.csv')  
figura10 = bokeh11('FicherosSalida/Clustering.csv')  
figura11 = bokeh14('FicherosSalida/Organizaciones.csv')
```

Observación:

Se han tenido problemas para cargar estos archivos en un fichero que luego se pueda

descargar. Esto es porque para que funcione un gráfico de Bokeh, necesita un Javascript y archivos de estilo css. Este soporte lo da el servidor, para que se pueda utilizar los gráficos sin el servidor hay que renderizar cada complemento en la página web.

```
js_resources = INLINE.render_js()
css_resources = INLINE.render_css()
```

El último tratamiento que hay que realizar sobre los gráficos es separar el script del gráfico y la parte html donde se va a cargar.

```
script2, div2 = components(figura2)
script3, div3 = components(figura3)
script4, div4 = components(figura4)
script5, div5 = components(figura5)
script6, div6 = components(figura6)

script7, div7 = components(figura7)
script8, div8 = components(figura8)
script9, div9 = components(figura10)
script13, div13 = components(figura13)
script11 , div11 = components(figura11)
```

Con esto la parte de los gráficos están preparados para uso en una página web, en el apartado de plantillas se explicará como se plasma cada gráfico en una página html.

Para darle sensatez al uso de esta herramienta, el usuario debe poder quedarse con los datos capturados además de con el análisis.

Para ello, cuando el usuario descargue el análisis, el archivo contendrá unas pestañas en el menú referentes a la captura de datos.

Para conseguir esta información, hay que pasar junto a los gráficos, las lista de datos descargados para crear en la página web unas tablas con estos contenidos.

La primera pestaña que tiene como nombre Captura, hace alusión a los datos de Twitter descargados, para esto le pasamos a la plantilla las siguientes listas:

```

fechatabla = data["fecha"].tolist()
tweets = data["full_text"].tolist()
idtabla = data["id_tweet"].tolist()
retweettabla = data["retweetCount"].tolist()
favoritetabla = data["FavoriteCount"].tolist()
verifiedtabla = data["userVerified"].tolist()
localizaciontabla = data["localizacion"].tolist()
nombretabla = data["NombreUsuario"].tolist()
URL = data["URLTweet"].tolist()

```

Index	Fecha	Full Text	ID Tweet	Retweet Count	Favorite Count	User Verified	Localización	Usuario	URL
1	2017-08-01 12:00:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234567	12	45	True	Madrid, España	@usuario1	https://twitter.com/usuario1/status/912345678901234567
2	2017-08-01 11:55:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234568	10	38	True	Barcelona, España	@usuario2	https://twitter.com/usuario2/status/912345678901234568
3	2017-08-01 11:50:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234569	8	25	True	Valencia, España	@usuario3	https://twitter.com/usuario3/status/912345678901234569
4	2017-08-01 11:45:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234570	5	15	True	Sevilla, España	@usuario4	https://twitter.com/usuario4/status/912345678901234570
5	2017-08-01 11:40:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234571	3	10	True	Zaragoza, España	@usuario5	https://twitter.com/usuario5/status/912345678901234571
6	2017-08-01 11:35:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234572	2	8	True	Málaga, España	@usuario6	https://twitter.com/usuario6/status/912345678901234572
7	2017-08-01 11:30:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234573	1	5	True	Bilbao, España	@usuario7	https://twitter.com/usuario7/status/912345678901234573
8	2017-08-01 11:25:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234574	0	3	True	Palma de Mallorca, España	@usuario8	https://twitter.com/usuario8/status/912345678901234574
9	2017-08-01 11:20:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234575	0	2	True	Las Palmas de Gran Canaria, España	@usuario9	https://twitter.com/usuario9/status/912345678901234575
10	2017-08-01 11:15:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234576	0	1	True	San Sebastián, España	@usuario10	https://twitter.com/usuario10/status/912345678901234576
11	2017-08-01 11:10:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234577	0	0	True	León, España	@usuario11	https://twitter.com/usuario11/status/912345678901234577
12	2017-08-01 11:05:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234578	0	0	True	Vitoria, España	@usuario12	https://twitter.com/usuario12/status/912345678901234578
13	2017-08-01 11:00:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234579	0	0	True	Gijón, España	@usuario13	https://twitter.com/usuario13/status/912345678901234579
14	2017-08-01 10:55:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234580	0	0	True	Astoria, España	@usuario14	https://twitter.com/usuario14/status/912345678901234580
15	2017-08-01 10:50:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234581	0	0	True	Valladolid, España	@usuario15	https://twitter.com/usuario15/status/912345678901234581
16	2017-08-01 10:45:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234582	0	0	True	Burgos, España	@usuario16	https://twitter.com/usuario16/status/912345678901234582
17	2017-08-01 10:40:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234583	0	0	True	Salamanca, España	@usuario17	https://twitter.com/usuario17/status/912345678901234583
18	2017-08-01 10:35:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234584	0	0	True	Castellón, España	@usuario18	https://twitter.com/usuario18/status/912345678901234584
19	2017-08-01 10:30:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234585	0	0	True	Tarazona, España	@usuario19	https://twitter.com/usuario19/status/912345678901234585
20	2017-08-01 10:25:00	¡Feliz cumpleaños! Te deseo un día maravilloso y un año lleno de felicidad. ¡Que todo salga bien!	912345678901234586	0	0	True	Teruel, España	@usuario20	https://twitter.com/usuario20/status/912345678901234586

Figura 10.3: Captura de tweets

Así, si el usuario en el caso de que quiera, puede copiar los valores. Siguiendo con la generación de texto para la página web se introduce el tweet más antiguo que se ha encontrado en la captura de tweets.

```

datafoco = pd.read_csv('FicherosSalida/Foco.csv', sep='\\t',
, engine="python")
textofofo = datafoco["full_text"].tolist()
usuariofofo = datafoco["NombreUsuario"].tolist()
urlfofo = datafoco["URLTweet"].tolist()
len4 = len(datafoco)

```

Hay que distinguir la parte texto, la parte del usuario y la url para que cuando se pinche en el texto se redirreccione a Twitter y el tamaño del fichero para que así en el caso de

haber muchos tweets los imprimamos en la página web.

Para mostrar los tweets que han escrito los usuarios, que hemos buscado relacionados con el tema de búsqueda, hay que realizar el mismo proceso.

```
data_usuario = pd.read_csv('FicherosSalida/TweetsUsuarios.csv',
    , sep='\t', engine="python", quotechar='"',
                                error_bad_lines=False)
tweets_usuario = data_usuario["full_text"].tolist()
usuarios_busqueda = data_usuario["NombreUsuario"].tolist()
url2 = data_usuario["URLTweet"].tolist()
len2 = len(data_usuario)
```

Para cada uno de los análisis, se ha creado un apartado en el menú, tanto para las localidades encontradas, hashtag y usuarios.

```
dataLocalidades = pd.read_csv('FicherosSalida/Localidades.csv',
    , sep='\t',
    , engine="python")
htmlLocalidades = dataLocalidades["localidad"].tolist()
htmlLocalidadesNumero = dataLocalidades["numero"].tolist()
lenLocalidad = len(htmlLocalidades)

dataHashtag = pd.read_csv('FicherosSalida/Hastag.csv', sep='\t',
    engine="python")
htmlHashtag = dataHashtag["Hashtag"].tolist()
htmlHashtagNumero = dataHashtag["numero"].tolist()
lenHashtag = len(htmlHashtag)

dataUsuarios = pd.read_csv('FicherosSalida/Usuarios.csv', sep='\t',
    , engine="python")
htmlUsuarios = dataUsuarios["Usuario"].tolist()
htmlUsuariosNumero = dataUsuarios["numero"].tolist()
lenUsuarios = len(htmlUsuarios)
```

Además hay que pasarle los análisis creados con los conjuntos de Wikipedia y de los valores que el usuario introdujo para realizar una subbúsqueda.

```
dataFichero = pd.read_csv('FicherosSalida/PuntuacionFichero.csv',
    sep='\t',
    engine="python")
htmlFichero = dataFichero["texto"].tolist()
htmlFicheroNumero = dataFichero["puntuacion"].tolist()
lenFichero = len(htmlFichero)

dataOrganizaciones = pd.read_csv('FicherosSalida/Organizaciones.csv',
    sep='\t',
    engine="python")
htmlOrganizaciones = dataOrganizaciones["Organizaciones"].tolist()
htmlOrganizacionesNumero = dataOrganizaciones["numero"].tolist()
lenOrganizaciones = len(htmlOrganizaciones)
```

Ahora hay que guardar la página que se va a crear para en el caso de que el usuario pinche en descargar este preparada.

Utilizando la función `render_template` y pasándole como atributos la plantilla, en este caso `resultadoDescarga.html`. Además del resto de variables antes nombradas, guardamos el fichero con el nombre del tema de búsqueda.

```
with open(""+tema+".html", "w") as file:
    file.write(paginaDescarga)
```

Por motivos de espacio en pantalla y para mejor compresión, se realiza otra página web que es la que se muestra al usuario conforme acaba la ejecución. Dicha página es más simple, solo contiene los datos de análisis y las gráficas.

Al tener el número mínimo de valores capturados de Twitter se manda esta plantilla `resultados2.html`, en el caso de no tener valores de Twitter o el usuario haya dado en el botón abortar nos redirecciona a la página principal del proyecto.

10.0.5. Plantillas creadas

En este apartado se va a explicar cada plantilla la función que tiene, que información muestra y su composición.

Flask nos brinda la capacidad de herencia entre plantillas para reutilizar código. Con este motivo se va a realizar una clasificación en una estarán los layout, son partes que heredan el resto de las plantillas y las propias plantillas.

Para heredar contenido de otra plantilla, esta plantilla hija debe incluir el contenido de la plantilla padre en su código:

```
{% extends "layout.html" %}
```

En este caso nuestra plantilla hija hereda de layout.html, en el caso de la plantilla padre layout.html hay que diferenciar el lugar donde se va incrustar el contenido de la plantilla hija, para esto se utiliza la función block content.

```
{% block content %}
```

```
{% endblock %}
```

El contenido de la plantilla hija será incrustado entre block content y endblock de la plantilla padre. A su vez el contenido que se quiera mostrar en la plantilla padre debe estar escrito entre estos parámetros para así realizar el enlazado.

```
{% block content %}
```

```

<style>
.egt {vertical-align: text-top;;}
</style>
<h2>Introducción</h2>

```

```
<table class="egt" >
```

```
<td valign="top" width="400" >
```

```

<p>En este TFG se va realizar un análisis de los tweets relacionados
con catastrofes: Inundaciones, incendios y accidentes.</p>

```

```
<p>Por medio de captura de datos de twitter, tanto en hashtag o
```

```

        búsqueda de palabras clave y
        realizando un seguimiento de los
        usuarios más importantes en estos
        ambitos.</p>

</td>
<td width="400">

    <a class="twitter-timeline" data-lang="es" data-width="500"
    data-height="700" href="
    https://twitter.com/IncendiosES?ref_src=twsrc%5Etfw">
    Tweets by IncendiosES</a>
    <script async src="https://platform.twitter.com/widgets.js"
    charset="utf-8"></script>

</td>

</table>

{% endblock %}

```



Figura 10.4: Pagina Principal

La parte azul que tiene el menú con las opciones introducción y captura, es la parte de

la plantilla padre layout.html. La parte de la derecha que tiene el texto de introducción y el widget de Twitter es la parte de la plantilla hijo index.html.

- Index.html:

Esta es la plantilla con la que se lanza el proyecto, es la primera que ve el usuario. En ella se muestran un texto con información sobre el proyecto, un widget de Twitter con tweets sobre catástrofes.

- Captura.html:

El contenido de esta plantilla es el formulario para los valores de entrada, aquí el usuario introduce el tema de búsqueda, el conjunto de valores para la subbúsqueda, todos estos separados por comas y el tiempo en segundos superior a 600.

Para la creación del formulario, hemos tenido en cuenta la entrada del usuario introduciendo restricciones:

- No poder introducir un campo vacío.
- El tiempo en segundos tiene que ser mayor o igual que 600.

Al cumplir estos valores el usuario pincha sobre el botón enviar para que se cargue la siguiente vista en este caso procesar.

```
<form method="POST" action="{{url_for('procesar')}}">
    <input name="tema" required type="text" size="35" placeholder="Tema de búsqueda">
    <br><br>
    <input name="valores" required type="text" size="35" placeholder="Valores de búsqueda">
    <br><br>
    <input name="tiempo" required type="number" size="20" min="600" placeholder="Tiempo en segundos">
    <br><br>
    <input type="submit" value="Enviar">
</form>
```

Esta plantilla hereda de la plantilla layout.html que contiene el menú de navegación de proyecto.

- **Espera.html:**

Esta plantilla es la que ve el usuario cuando se está realizando la captura de tweets y el análisis de los datos.

El contenido de esta plantilla es un resumen de los datos introducidos previamente, dos botones comenzar y abortar. Estos botones se diseñaron para ayudar al desarrollador a realizar pruebas con Flask. En el caso de realizar una herramienta final fuera de este TFG, habría que quitarlos, además de poner un sistema de alertas en el caso de que el usuario abandone el experimento. No se han contemplado por ser una web de pruebas que su única función es probar si el TFG era posible su realización.

El usuario al pinchar en comenzar empieza la captura de tweets y seguidamente su análisis.

Resumen de valores introducidos:

tema para la búsqueda general: COVID

valores para la subbúsqueda: Sanidad,Hospitales,China,contagios,muertes,casos,contagio

tiempo de captura: 600

Estado:

Numero de tweet descargados:

Figura 10.5: Pagina de espera y captura de tweets

Para hacer más llevadera la espera del usuario, se ha incorporado una variable que indica el estado de la ejecución, un contador con el número de tweets descargados y un área de texto el cual contiene los tweets que se están descargando.

Ahora se va realizar un análisis de las funciones que se han creado para esta vista:

- Botón Comenzar:



Figura 10.6: Pagina de descarga en ejecución

Para que cuando el usuario pinche en el botón el servidor lance la ejecución de la captura hay que realizar una llamada al controlador desde una vista.

```
<form method="POST" action="{{url_for('Capturar')}}">

onsubmit="myFunction()">

<input id="comenzar" type="submit" value="Comenzar">

</form>
```

El formulario tiene un método del tipo POST porque se va a enviar información a la próxima vista, esta vista es Capturar.

Una observación puede ser que la sintaxis con la que se envía a otra vista es con las llaves y utilizando la función de flask `url_for`.

Para dar un poco más de seguridad a la aplicación, el botón comenzar se bloquea tras ser pulsado, esto es para que el usuario pulse solo una vez.

Para realizar esta funcionalidad se ha implementado en javascript el siguiente código:

```
<script>
function myFunction() {
```

```

        document.getElementById("comenzar").disabled = true;
    }
</script>

```

Lo que se realiza en la función es la deshabilitar el botón comenzar.

- Botón abortar:

Para la funcionalidad del botón abortar se realiza de la misma forma que en el Comenzar.

```

<form id="abortar" method="POST" action="{url_for('abortar')}">

    <input type="submit" value="Abortar">

</form>

```

- Contador de tweets:

Para que se actualice el número de tweets que se va descargando tenemos que realizar una petición al controlador del servidor. La petición va sobre una función que nos retorna el valor y lo plasmamos en el código html.

Para realizar esta función se han utilizado dos lenguajes de programación web Javascript y Ajax.

```

<script type="text/javascript">
    window.setInterval(function(){
        $("#update").load("/contador_tweets?");
    }, 10000)
</script>

```

Se llama a la vista contador_tweets y la vista lanza la función actualizaContador, esta devuelve un valor numérico convertido a texto e incluido en un párrafo html. La función se ejecuta periódicamente cada 10 segundos y actualiza el objeto con el id update.

```

@app.route("/contador_tweets")
def actualizaContador():
    a= get_contador()
    return "<p> " + str(a) + " </p>"

```

- Estado de ejecución:

Para que el usuario tenga una información sobre el estado de la ejecución existe esta variable Estado, para su implementación se ha recurrido a javascript y ajax.

```
<script type="text/javascript">
    window.setInterval(function(){
        $("#update2").load("/Estado?");
    }, 10000)
</script>
```

Este script llama cada 10 segundos al controlador y este ejecuta la vista Estado, en esta vista se lanza la función actualizaEstado.

```
@app.route("/Estado")
def actualizaEstado():
    global Estado
    return "<p> " + str(Estado) + " </p>"
```

Al igual que en la función actualizaContador tenemos que devolver el texto en un párrafo html, en este caso el valor que se va a actualizar es update2.

- Tweets por pantalla:

La forma que se ha utilizado para mostrar los tweets por pantalla es otra forma de aumentar el feedback con el usuario, para que pueda ir leyendo los tweets mientras que la captura y el análisis finalizan. Para este script se ha utilizado Javascript, Ajax y Json.

```
<script type="text/javascript">
window.setInterval(function(){
    var targ = $("#w3mission").load("/tweets");
    document.getElementById("w3mission").innerHTML += JSON.stringify(targ);
}, 10000)
</script>
```

Esta función se repite cada 10 segundos, se llama a la vista tweets y esta lanza actualizaTweets. Una vez que esta función nos devuelve un string con los tweets que se han capturado hasta ahora, lo guardamos en la variable targ. El problema

de pasar directamente el contenido de la variable al textArea es que da un valor indefinido, esto es por que hay que reconvertir el texto para que se pueda representar en el textArea. Para ello nos apoyamos en la función stringify, de la biblioteca de JSON y por último lo convertimos en html.

■ Resultados2:

Esta plantilla es la que el usuario va a ver al terminar la ejecución del análisis, se va a explicar la forma de colocar cada gráfica y cada texto generado en el análisis en esta plantilla.

- Cargar una línea de texto:

La forma de cargar texto simple en la página web, es por medio de dobles llaves y el contenido seguido por el parámetro safe.

```
{{ tema|safe }}
```

En este ejemplo se está plasmando en la página de los resultados la variable tema que contiene el tema de búsqueda que introdujo el usuario. Los valores de búsqueda que introdujo el usuario y el tiempo en segundos se introduce de esta manera.

- Mostrar una lista:

Para mostrar una lista tenemos que recorrer esta, imprimirla y plasmarla en código html.

```
<table id="tabla" style="width:35%" border="1">
  <tr>
    <th>Numero</th>
    <th>Tweet</th>
  </tr>
  {%for i in range(0, len)%}
  <tr>
    <td width="10%" align="center">{{i+1}}</td>

    <td width="30%"><a href={{url[i]}}>{{tw[i]}}</a></td>

  </tr>
  {%endfor%}
</table>
```


En la pestaña de análisis de la captura se muestra la captura de texto de los tweets. Se ha mostrado en una tabla html, esta contiene una primera fila con las cabeceras número y tweet, para ahora mostrar la lista de los tweets contenidos en la variable tw, es necesario recorrerla con un bucle for y de rango hasta el final de la lista, para plasmar el texto en html es necesario el uso de dobles llaves.

- Gráficos bokeh:

Para los gráficos hechos con Bokeh hay que cargar las dos partes de cada gráfica el Script y el propio gráfico.

Primero hay que cargar en el html los archivos renderizados de javascript y hojas de estilo.

```
{{ js_resources|indent(4)|safe }}
{{ css_resources|indent(4)|safe }}
```

Ahora añadimos a la página web la parte Script del gráfico.

```
{{ plot_script2|indent(4)|safe }}
```

Y por último ya el propio gráfico.

```
<div>{{ plot_div4|indent(4)|safe }}</div>
```

Vamos a explicar la organización de esta página web, para que así resulte más sencillo entenderlo. Para diferenciar cada parte del análisis creado se ha utilizado un menú horizontal con los campos de cada análisis. Esta versión es para la vista que el usuario ve todavía estando en el servidor. Si el usuario pincha en descargar, la página web que descargue tendrá además los apartados de las capturas de tweet. El motivo de realizar dos vistas distintas para los resultados es por el espacio en pantalla, debido a que cuando el usuario descargue el archivo no tendrá las redirecciones del servidor flask, será un archivo html por lo cual debe estar toda la información en la misma ventana.

Analisis de eventos en twitter	
Analisis de la captura	
Resumen de valores de entrada:	
Numero de tweets 2307	
Valor de entrada para la busqueda general: Betis	
Valores para la subbusqueda: Villarreal,liga	
Tiempo en segundos de la captura de tweets: 600	
Medio de descargas del algoritmo de tiempo real: 31.5	
Numero medio de descargas del Busqueda: 599.5	
Numero medio de descargas de Busqueda2: 1000	
Numero medio de descarga total: 543.6666666666666	
Numero	Tweet
1	Finalizo el encuentro Betis - Villarreal por la jornada 33 de #LaLiga. El submarino amarillo desplego un gran juego ante un Betis nuevamente perdido y sin profundidad. #LaLigaSantander
2	Que el Betis le gane al celta el proximo partido, le haga el favor al mallorca, y acabar la temporada
3	El Villarreal le ha podido meter 6 al Betis
4	La sensacion de que, si llega a estar un poco mas acertado de cara a puerta, el Villarreal le mete 6 al Betis
5	@Richy30926581 @RealBetis @Guido_Rodriguez_Si, pero betis ni llega a la ligulla digo. Quien habla de Mexico jajaja. El pasado es el pasado. Con esta defensa ni se acuerdan en la primera. Ay se van dando con el pueblo pensando por no defender
6	No soy capaz de explicarte a mi hijo que los problemas del Betis son culpa de los arbitros. A Mateo y a alguno igual les sale como resorte, sera por amar al Betis?
7	@Jus785FC Fokir quiso ir al Betis por su mazapota y por la grandeta que le sale de no se donde
8	@rrraullititos Creeo q no existe jajaja o por lo menos no quiere al Betis... v normal
9	Totalmente de acuerdo, esto es un desproposito
10	Como es que el Betis no esta metido en plena lucha por el descenso?
11	Min. 91 (0-2): Lyl! Disparo lejano de Ontiveros que tuvo que enviar Joel Robles a saque de esquina #AFDPLive
12	Y mira, en el minuto 91 del partido, despues de traer un 4-2 y un 0-2, de ver como los arbitros se descontrolan en nuestra cara y contemplar (al igual que lo hace Joel) como nos llueven los goles, creo que va siendo hora de que la

Figura 10.7: Pagina de resultados

Además se le añadió a la página web un menú retráctil para aprovechar al máximo el espacio de la pantalla.

Analisis de eventos en twitter	
Cerrar x	
Analisis de la captura	
Analisis Tweets	Analisis de la captura
Analisis de Diccionarios	Resumen de valores de entrada:
Analisis Usuarios	Numero de tweets 2307
Agrupamiento	Valor de entrada para la busqueda general: Betis
Captura v	Valores para la subbusqueda: Villarreal,liga
	Tiempo en segundos de la captura de tweets: 600
	Medio de descargas del algoritmo de tiempo real: 31.5
	Numero medio de descargas del Busqueda: 599.5
	Numero medio de descargas de Busqueda2: 1000
	Numero medio de descarga total: 543.6666666666666
Numero	Tweet
1	Finalizo el encuentro Betis - Villarreal por la jornada 33 de #LaLiga. El submarino amarillo desplego un gran juego ante un Betis nue #LaLigaSantander
2	Que el Betis le gane al celta el proximo partido, le haga el favor al mallorca, y acabar la temporada
3	El Villarreal le ha podido meter 6 al Betis
4	La sensacion de que, si llega a estar un poco mas acertado de cara a puerta, el Villarreal le mete 6 al Betis
5	@Richy30926581 @RealBetis @Guido_Rodriguez_Si, pero betis ni llega a la ligulla digo. Quien habla de Mexico jajaja. El pasado es, acuerdan en la primera. Ay se van dando con el pueblo pensando por no defender
6	No soy capaz de explicarte a mi hijo que los problemas del Betis son culpa de los arbitros. A Mateo y a alguno igual les sale como re
7	@Jus785FC Fokir quiso ir al Betis por su mazapota y por la grandeta que le sale de no se donde
8	@rrraullititos Creeo q no existe jajaja o por lo menos no quiere al Betis... v normal
9	Totalmente de acuerdo, esto es un desproposito
10	Como es que el Betis no esta metido en plena lucha por el descenso?
11	Min. 91 (0-2): Lyl! Disparo lejano de Ontiveros que tuvo que enviar Joel Robles a saque de esquina #AFDPLive
12	Y mira, en el minuto 91 del partido, despues de traer un 4-2 y un 0-2, de ver como los arbitros se descontrolan en nuestra cara y c como nos llueven los goles, creo que va siendo hora de que la directiva del Betis se replantee un poco todo.

Figura 10.8: Menú desplegado

- Análisis de la captura:

Aquí en esta pestaña se muestran los valores que introdujo el usuario, el número de tweets descargados, el tema de búsqueda general, los valores para la subbúsqueda de valores y el tiempo de captura de datos.

Para dar un análisis estadístico a la captura se muestra los valores medios conseguidos por cada algoritmo y la captura media de valores entre los tres algoritmos de captura de tweets. Los datos anteriores son datos resumen, abajo se muestra una tabla con el texto de cada tweet descargado, como funcionalidad

añadida al pinchar en el texto redirrecciona al tweet en Twitter.

- Análisis Tweet:

En esta pestaña se muestra la parte más social de la captura y las entidades encontradas.

Lo primero con lo que nos encontramos es el tweet más antiguo que se ha encontrado y su usuario. Se le añadido la funcionalidad de que al pinchar se redirreccione a dicho tweet.

A continuación expone el número de tweets que son de usuarios verificados y el número de usuarios estándar.

El primer gráfico que se muestra es la el número de tweet por instante de tiempo que se ha capturado.

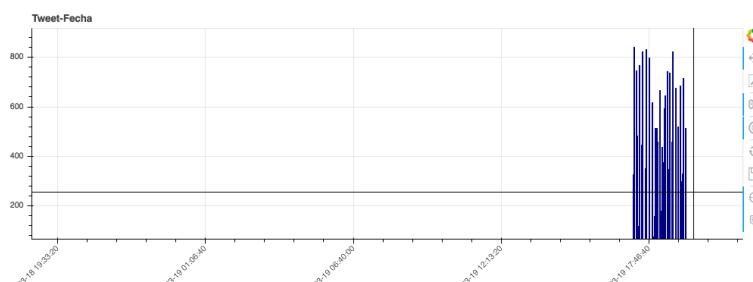


Figura 10.9: Gráfica Bokeh tweets por fecha

El siguiente gráfico a mostrar en este apartado, es el que relaciona los tweets con el número de retweet obtenido.

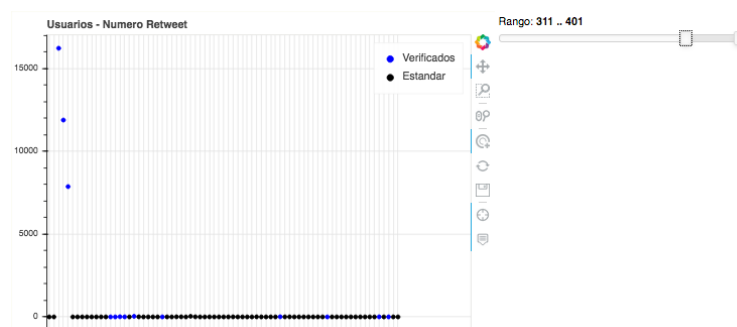


Figura 10.10: Gráfica de la captura de retweets

El gráfico de los usuarios por el número de favoritos es el siguiente, en este grafico se relacionan los usuarios de cada tweet y el número de favoritos que ha tenido ese tweet.



Figura 10.11: Ejemplo de tweet

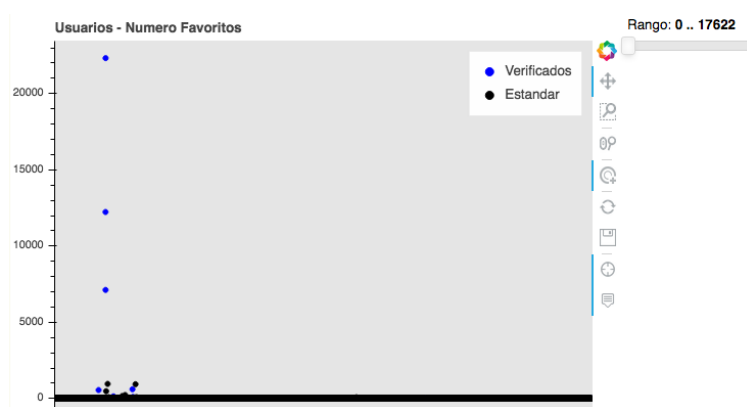


Figura 10.12: Gráfica de retweets

El gráfico de Puntuación - Hashtag es para resaltar los hashtag encontrados en los

tweets y ver cuales han tenido más relevancia en estos.



Figura 10.13: Ejemplo de gráfica con la captura de hashtag

Los usuarios más nombrados se pueden ver en el gráfico Puntuación - Usuarios.

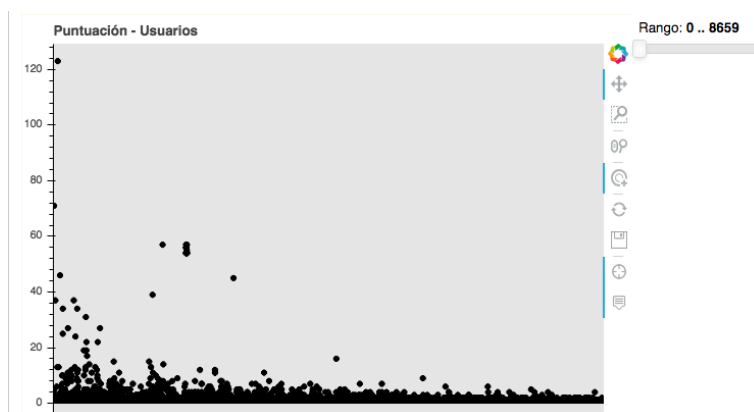


Figura 10.14: Ejemplo de gráfica de usuarios

Los dos últimos gráficos de este apartado son el gráfico de localidades en el que se muestran las localidades encontradas y el número de veces.

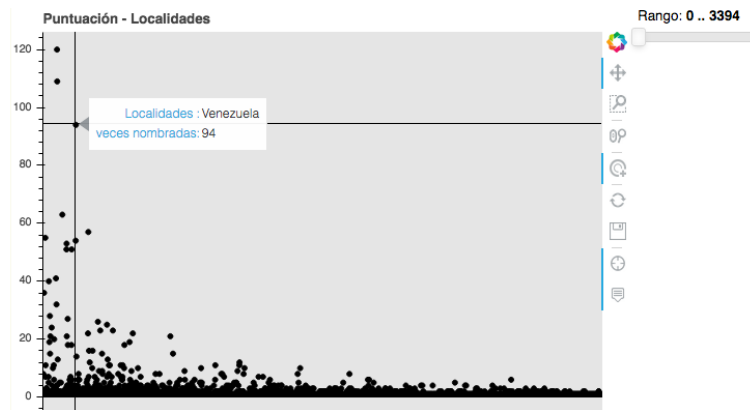


Figura 10.15: Gráfica localidades

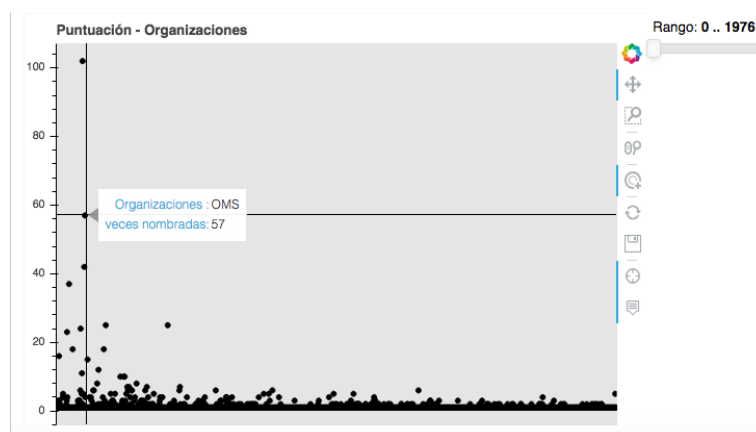


Figura 10.16: Gráfica de organizaciones

- Análisis de usuarios:

En este apartado de resultados.html se muestra los usuarios encontrados sobre el tema de búsqueda y los tweets de cada uno relacionados con el tema de búsqueda.

Cada texto es una enlace hacia el tweet fuente en Twitter.

- Análisis Diccionarios:

Aquí se muestran las palabras relacionadas con el tema de búsqueda que se han encontrado en Wikipedia y además también se muestran los valores escritos por el usuario, los valores están lematizados. Seguido de los valores de cada conjunto, se muestran las gráficas de datos de cada uno.

Analisis Usuarios

Usuarios encontrados sobre el tema: COVID

['COVIDNewsByMIB', 'Cloud9', 'UtahCoronavirus', 'OmanVSCovid19', 'MantralayaRoom', 'COVID_Australia', 'floydimus', '2019nCoVwatcher', 'coronavirusscare', 'covidperspectiv', 'DigiCommsNG', 'idubbbz', 'COVID2019tests', 'COVID19_USA_', 'Wakazi', 'CarlosR', 'COVID19Tracking', 'DIPR_COVID19', 'CovidAidUK', 'SEACoronavirus']

Usuario	Tweet
COVIDNewsByMIB	RT @fssaiindia: It is everyones responsibility to take the right actions in order to help fight #COVID2019. #HealthForAll #IndiaFightsCor
UtahCoronavirus	@ReidJohnFuller If you think you've been exposed to COVID-19 and develop a fever and symptoms (cough, difficulty breathing), call your healthcare provider for medical advice. In the meantime, practice all the recommended measures of social distancing.
OmanVSCovid19	#_#19_Coronavirus #Covid19 infection & recovery cases updates. #_#
COVIDNewsByMIB	#IndiaFightsCorona Domestic cargo flights strengthen Indias fight against #COVID19; medical supplies-testing kits, masks, gloves and more being delivered across the country since 26th March 2020:
COVIDNewsByMIB	RT @moayush: Arogya Setu App - Stay informed with the latest updates on the fight against COVID-19. The Government has launched a Bluetoot
COVIDNewsByMIB	LIVE SHORTLY: Media briefing on current Novel #Coronavirus situation in the country at National Media Centre in #NewDelhi Time: 4:00 PM #IndiaFightsCorona #COVID19Pandemic @MoHFW_INDIA @PIB_India

Figura 10.17: Análisis de los usuarios



Figura 10.18: Gráfica de analisis Wikipedia

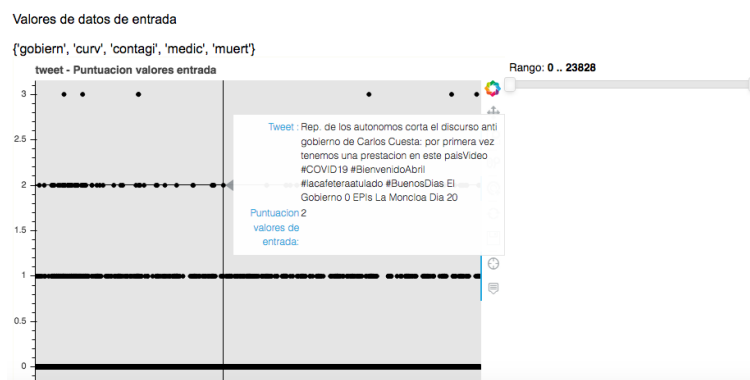


Figura 10.19: Gráfica con los valores introducidos por el usuario

- Agrupamiento:

La pestaña agrupamiento muestra el número de tweets descargados y se muestra el número de tweets escogidos para realizar entrenamiento de K-means. Se muestra un gráfico con la representación del método del codo.

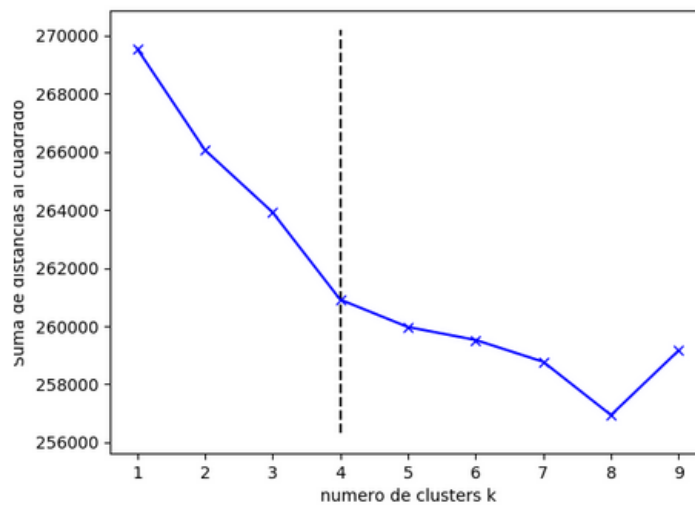


Figura 10.20: Gráfica ejemplo método del codo

Como se menciona antes, aquí se introduce los centroides más relevantes de cada cluster.

Los centroides mas representativos del conjunto de datos son los siguientes:

cluster 0,nan,zzu,entrado,entree,entre,entraste,entras,entraron,entrare,entrar

cluster 1,the,to,of,and,is,in,covid,for,19,this

cluster 2,covid,19,covid2019,coronavirus,mas,covid19,si,covid_19,via,casos

Figura 10.21: Centroides más representativos

Y por último en este apartado se representan las gráficas de los cluster separados por color.

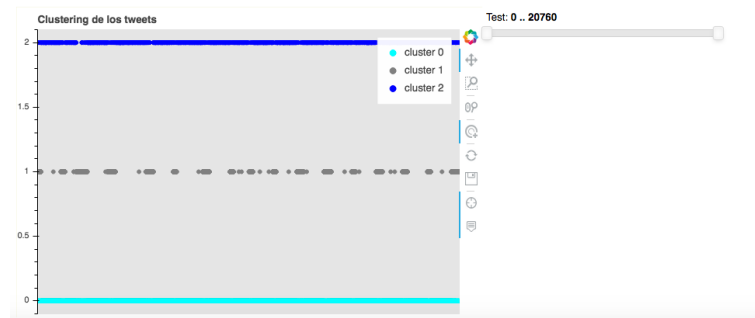


Figura 10.22: Gráfica de agrupamiento

■ ResultadosDescarga.html:

Esta plantilla es la que el usuario se descargue para su uso personal, conteniendo la información de la plantilla resultado2.html. Además en esta plantilla se han introducido la captura de tweets y los análisis de cada parte.

- Análisis Captura
- Análisis tweet
- Análisis Usuarios
- Análisis Diccionarios
- Agrupamiento
- Captura:
Este apartado hace referencia a la captura de los tweets con los campos estudiados en durante el proyecto.
- Captura localidades:
Son el nombre de cada localidad encontrada y el número de veces que ha salido en los tweets.
- Captura Usuarios:
Muestra una tabla con el nombre de los usuarios y la veces que han sido nombrados en los tweets.

- Captura Hashtag:
El nombre de los hashtag utilizados en los tweets y el número de veces que han sido escritos.
- Captura Wikipedia:
Cada tweet y la puntuación en base a las veces que una de las palabras del conjunto de Wikipedia han salido.
- Captura de valores de entrada:
Cada tweet y su puntuación en base a las palabras encontradas del conjunto de valores introducidos por el usuario.
- Captura Organizaciones:
Una tabla con las organizaciones encontradas y el número de veces encontradas en los tweets.

El principal problema era como organizar tanta información en una página web para que no se mezclara cada apartado, porque al no tener un servidor que realizara las redirecciones entre páginas. Todo la información tiene que estar en la misma página, si se hubiera tomado la idea de ponerlo todo seguido, el usuario en consecuencia estaría subiendo y bajando la página para poder analizar la información. Para solucionar este problema se utiliza un menú que redirrecciona dentro del propio html, así la página está organizada por secciones y el usuario tiene una forma más fácil su comprensión.

```
<div class="tab">
<button class="tablinks" onclick="openCity(event, 'Análisis Estadístico')> id
Análisis de la captura</button>
<button class="tablinks" onclick="openCity(event, 'Análisis Tweet')">
Análisis Tweet</button>
<button class="tablinks" onclick="openCity(event, 'Análisis Usuarios')">
Análisis Usuarios</button>
<button class="tablinks" onclick="openCity(event, 'Análisis Diccionarios')">
Análisis Diccionarios</button>
<button class="tablinks" onclick="openCity(event, 'Clustering')">
Agrupamiento</button>
<button class="tablinks" onclick="openCity(event, 'Completo')">
Captura</button>
<button class="tablinks" onclick="openCity(event, 'CompletoLocalidades')">
Captura Localidades</button>
<button class="tablinks" onclick="openCity(event, 'CompletoUsuarios')">
```

```

Captura Usuarios</button>
<button class="tablinks" onclick="openCity(event, 'CompletoHashtag')">
Captura Hashtag</button>
<button class="tablinks" onclick="openCity(event, 'CompletoWiki')">
Captura Wikipedia</button>
<button class="tablinks" onclick="openCity(event, 'CompletoFichero')">
Captura valores entrada</button>
<button class="tablinks" onclick="openCity(event, 'CompletoOrganizaciones')">
Captura organizaciones</button>
</div>

```

Un problema que tiene este archivo es que tarda en cargarse por el tamaño. El programa que tiene que cargarlo son los navegadores web, al ser el lenguaje html un lenguaje interpretado tiene que cargarse integro en la memoria. Esto supone un tiempo de espera por parte del usuario. Otro inconveniente que tenía es que cuando se cargaba el archivo no se mostraba nada hasta que el usuario no pinchaba. Para solucionar esto, cuando se carga la página, la primera página en mostrarse es la de análisis de la captura. Esto se ha conseguido por un script en Javascript, que realiza el acto clicar en la pestaña análisis de la captura.

```

<script>
function openCity(evt, cityName) {
    var i, tabcontent, tablinks;
    tabcontent = document.getElementsByClassName("tabcontent");
    for (i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = "none";
    }
    tablinks = document.getElementsByClassName("tablinks");
    for (i = 0; i < tablinks.length; i++) {
        tablinks[i].className = tablinks[i].className.replace(" active", "");
    }
    document.getElementById(cityName).style.display = "block";
    evt.currentTarget.className += " active";
}

// Get the element with id="defaultOpen" and click on it
document.getElementById("defaultOpen").click();

```

```
</script>
```

- Layout.html:

Esta plantilla las heredan la siguientes plantillas: index.html, captura.html. En ella se muestra un menú de navegación con las opciones de Introducción y de captura, esto es para que el usuario navegue por las vistas del proyecto.

Para redireccionar cada enlace del menú hay que utilizar dobles llaves y url_for con el nombre de la función, para que la vista llame al controlador y este ejecute la función.

```
<div class="w3-bar-block">
  <a href="{{ url_for('home') }}" onclick="w3_close()" class="w3-bar-
    item w3-button w3-hover-white">Introducción</a>
  <a href="{{ url_for('captura') }}" onclick="w3_close()" class="w3-bar-
    item w3-button w3-hover-white">Captura</a>

</div>
```

La primera opción del menú redirecciona a la vista home esta envía la plantilla index.html antes nombrada, la opción captura hace referencia a la vista captura.html.

- Layout2.html:

Este tiene el mismo contenido que layout.html, lo único que lo diferencia es que tiene las opciones quitadas de navegación. Esto se hace porque al no haber en el servidor caché, hay que quitar la opción de volver al index.html mientras se realiza una captura.

En el caso de que hubiera gestión de caché en el servidor se habilitarían la opciones de introducción y captura cuando se lanza un experimento. En el apartado de conclusiones se explicará que cambios tendrían que realizarse para pasar este proyecto a una aplicación final.

10.0.6. Variables

Se van explicar las variables del controlador.

- App: Es la instancia del servidor flask.
- Tres: contador del intervalo de tiempo para lanzarlo cada 3 minutos.
- Cinco: contador del intervalo de tiempo para lanzarlo cada 5 minutos.
- Nueve: contador del intervalo de tiempo para lanzarlo cada 9 minutos.
- Tema: búsqueda general introducida por el usuario.
- Tiempo2: variable de apoyo
- Valores: valores introducidos por el usuario para la búsqueda en el conjunto de datos.
- Estado: variable con actualización del estado de la ejecución.
- Tiempoejecucion: variable con la que se compara el límite de tiempo que ha introducido el usuario.
- Limit: límite de tiempo para la captura de datos.
- PaginaDescarga: variable contenedora de la página que va a descargar el usuario.
- Tweets: variable donde se van a guardar los tweets que se van a mostrar mientras el usuario espera.
- ListaBusqueda: lista con el número de tweets, que ha descargado la función Busqueda.
- ListaBusqueda2: lista con el número de tweets que ha descargado la función Busqueda2.
- ListaTiempoReal: lista con el número de tweets que ha descargado CapturaTiempoReal.

Capítulo 11

Conclusiones

Después de la realización de este TFG se va escribir sobre las conclusiones que se han realizado sobre el proyecto.

- Usabilidad del proyecto:

El proyecto tiene una gran utilidad debido a que se puede extrapolar a varios ámbitos, no solo a catástrofes. En las primeras pruebas se utilizaron como sujeto de estudio eventos sociales como: partidos de fútbol, conciertos de musica, apertura de betas de juegos y opinión sobre películas.

El poder tener un análisis de los textos publicados por los usuarios de Twitter, es una herramienta para saber la opinión de un gran número de personas.

La generación de gráficas las cuales un analista puede utilizar para filtrar los datos, hacer énfasis en los usuarios que son verificados para realizar un seguimiento sobre estos y dan versatilidad al análisis por parte del usuario.

- Observaciones sobre el proyecto:

En el proyecto se han tenido que realizar algunas decisiones de implementación, las cuales podrían ser otras muy diferentes como por ejemplo:

- El lenguaje de programación:

El usar Python como lenguaje de programación ha hecho que el alumno rápidamente avance en muchos de los aspectos del proyecto, pero por ejemplo utilizar un lenguaje como c++ daría más velocidad de computo a la hora de ejecutar el proyecto. También el usuario hubiera tenido más dificultades a la hora de utilizar

las librerías de agrupamiento, actualmente Python es el lenguaje más utilizado en el ámbito de la investigación.

- Las claves de acceso a Twitter:

El tener total acceso a Twitter también habría sido un factor muy relevante a la hora de realizar el proyecto (menos tiempo de espera, más utilidades de la Api, se podría haber realizado un análisis de cada usuario mencionado en cada tweet, viendo características como si es verificado, realizar un análisis de la descripción de la cuenta de Twitter). El usar unas claves de Api profesionales y el lanzar más ejecuciones simultáneas, tendría como negativo realizar métodos de programación concurrente para el acceso a los archivos contenedores.

- La interface de usuario:

El tener una interface de usuario web, hace que la aplicación si en un futuro se subiera a un servidor tendría mucha más accesibilidad en cuanto al tipo de dispositivos electrónicos la pudieran usar.

Esto tendría el inconveniente de realizar unas vistas y unos archivos finales adaptados a estos (tablets y smartphones). En las primeras versiones se realizó una interfaz de escritorio, lo que hubiera dado problemas al insertar los gráficos bokeh por esto y se cambió a web.

- Experimentos realizados:

El número de experimentos de captura de datos han sido 16 y el número de experimentos con salida como archivo tipo html han sido 33. Los archivos generados tienen una media de tamaño de 20 megas, todo esto depende del número de datos que se captura. La velocidad de apertura en el equipo en el que se ha realizado el proyecto ha sido de media 15 segundos. Para una futura aplicación podría tenerse en cuenta la realización registro de usuarios y que en un servidor se guardara la capturas de datos, para así el usuario lanza el experimento y no tiene porque quedarse viendo como termina. En las primeras versiones de captura, dejaba el ordenador capturando datos y programaba el envío de un correo con un archivo de texto, con los datos de la captura, indicando el número de tweets escritos. Al ser los archivos de un tamaño superior a 10 megas solo podía enviar resúmenes de las capturas.

- Prueba de uso con varias personas:

La última semana en la que se ha realizado los últimos cambios en el proyecto se eligió a varias personas para el uso del proyecto para dar una opinión sobre este.

- Primer individuo:

Este individuo está cursando la carrera de grado en ingeniería informática.

Observaciones:

El proyecto funciona bien, he utilizado como sujeto de búsqueda al youtuber Dalas, introduciendo como subvalores: feminismo y mujer.

Como parámetro de tiempo se introdujo 600 segundos el tiempo mínimo, me sorprendió que en ese tiempo capturará 2780 tweets diferentes en teoría, el poder pinchar en cada texto para la redirección a Twitter es útil para ver la fuente del tweet.

Los gráficos resaltan los usuarios más relevantes en la captura, además de que yo solo puse Dalas, pero en la búsqueda de Twitter hay tweets en los que se ve el hashtag DalasIsOverParty, al contener el hashtag la palabra Dalas también realiza una búsqueda sobre este.

Cosas que mejoraría:

He estado tiempo esperando sin saber que estaba pasando, no sabía si se había bloqueado el programa o directamente no funcionaba y los colores en la gráfica, el rojo simboliza peligro no es el más idóneo.

- Segundo individuo:

Estudiante de máster de Mecánica, la aplicación es hiperútil y yo la utilizaría sin duda para recalcar datos de usuarios. Un ejemplo para el torneo de Interautonómico de Velocidad, aquí podríamos ver que tipos de usuarios están siguiendo el torneo, en las palabras clave pondría mi equipo de la universidad de Jaén.

- Tercer individuo:

Médico de 53 años, este individuo realizó un análisis sobre el coronavirus, sus opinión es la siguiente:

Para un médico encontrar todos los usuarios especializados sobre el tema del coronavirus puede ser muy útil, la pestaña de análisis de los tweet desde un aspecto más social. Esto es de menos relevancia, pero las graficas con las organizaciones que han estado en la captura de los textos si es importante, ver que opina el gobierno de la nefasta administración de los recursos públicos. Otro uso que aunque no está en mi profesión sería para las fuerzas de seguridad del estado ya que pueden detectar personas que se salten la cuarentena.

Cosas que mejoraría: hay muchas pestañas del análisis que no entiendo su uso, debido a mi falta en conocimientos informáticos, para profesionales que no son del ámbito de la informática necesitarían un curso de iniciación a este tipo de tecnologías.

En resumen los usuarios han tenido una buena experiencia con la aplicación del proyecto,

pero debido a que cada uno es de ámbitos profesionales distintos, tienen distintas visiones sobre la aplicación y sus futuros usos.

■ Paso a una aplicación final:

Para pasar a una aplicación final, se tendrían que realizar las siguientes modificaciones en el proyecto:

- Realizar vistas para el uso en diferentes sistemas electrónicos.
- Cambiar las claves de acceso a la Api, para evitar esperas y ganar más datos capturados en menos tiempo.
- Crear un sistema de registros de usuarios y de almacenaje de las capturas en un servidor.
- Realizar mejoras en el servidor para que no haya que estar esperando mientras que se lanza el experimento, un sistema de colas de experimentos con alertas al usuario con su finalización.
- Un órgano que supervise los pagos de sistema de almacenaje en la nube y de los pagos por las claves de alto rendimiento de Twitter.

■ Añadir otras funcionalidades:

Otras funcionalidades que le darían más riqueza al análisis serían el uso de otras redes sociales, como Facebook debido al gran número de páginas que tienen un seguimiento a incendios y catástrofes naturales.

Aplicar inteligencia artificial para analizar los usuarios mencionados en los tweets, para clasificarlos por relevancia con el tema de búsqueda.

■ Opinión del Alumno:

Mi experiencia sobre este TFG ha tenido varios tipos de momentos, en los cuales sufría por el hecho de no saber si podría darle solución al estudio. Esto es debido a que mi tutor me lo advirtió antes de empezar el proyecto, buscamos si es factible la realización de este tipo de programas a lo mejor nos toca cambiarlo todo a mitad del curso, por suerte y por las horas que le he dedicado al TFG se ha llegado una solución.

Luego no todo ha sido malo, he aprendido mucho sobre algoritmos que en mi rama de la carrera no los había aprendido, al uso de herramientas tanto de análisis, como de generación de documentos que ni siquiera sabía que existieran.

Por último, quiero dar las gracias a mi tutor Manuel García Vega por sugerir este trabajo de fin de grado, por el gran número de tutorías, todas las veces que me ha guiado y lo que me ha aportado.

Ya que este trabajo de fin de grado me ha hecho aprender multitud de herramientas y de lenguajes de programación que seguro me servirán para mi futuro profesional.

Capítulo 12

Conocimientos adquiridos por el alumno

El usuario ha cursado el grado en ingeniería informática en la rama de Tecnologías de la información. Los conocimientos que he aprendido en el transcurso del trabajo del fin de grado son los siguientes:

- Conocimientos en Sistemas de recuperación de información.
- Procesamiento de lenguaje natural.
- Conocimientos de la API de Twitter y de su wrapper Tweepy.
- Manipulación de los datos en formato JSON
- Creación de páginas web con el microframework Flask.
- Conocimientos en la librería Bokeh, para la creación de gráficos interactivos.
- Introducción al uso de redes neuronales.
- Uso biblioteca Nltk, Sklearn y Spacy.
- Introducción de herramientas de agrupamiento de datos.
- Aprendizaje en el uso de Látex para crear la documentación del trabajo de fin de grado.
- Introducción a Ajax.
- Introducción a Javascript.

Capítulo 13

Trabajos futuros

Aquí se va a plantear distintos tipos de estudio y un pequeño análisis sobre como lo aprendido en este TFG podría ayudar en estas investigaciones.

- Varios agentes de búsqueda y análisis

En el TFG he utilizado solo un ordenador para realizar las capturas de datos de Twitter, sería más eficiente utilizar varios equipos cada uno con un conjunto de claves de desarrollador distintas, con esto mejoraría los tiempos parados debido a los límites que nos da la API. El sistema tendría un servidor central el cual el usuario analista insertaría su consulta, esta se transpasaría a cada uno de los nodos que realizaría una búsqueda, después al terminar ya fuera por una condición de tiempo o por un límite de tamaño estos devolverían un Json con la descarga de datos. Incluso otro aspecto que podría mejorar esta configuración es el uso de distintos tipos de búsqueda. Ejemplo:

Si el usuario nos introduce incendio en Jaén, el servidor le puede mandar a un nodo que investigue en una determinada fecha, a otro en otros lugares.

- creación de un tesoro con palabras relacionadas

Este es un tema importante realizar un buen tesoro para realizar consultas, Wikipedia puede ser un pequeño paso pero siempre con un análisis por medio de una persona, la Api de Wikipedia suele meter valores inconcluso que no tiene nada que ver con el tema de investigación.

- Utilización de distintas versiones de API Rest

Realizar el mismo trabajo de investigación que se ha realizado pero utilizando tras diferentes claves de Twitter, cada clave tiene unas funcionalidades distintas como el periodo de búsqueda, o el límite de descargas de tweet.

- Herramienta no solo en España

Para este TFG, la pequeña aplicación que se ha realizado esta enfocada en España y tweets de aquí, sería interesante extrapolarlo a otros países.

- Exportar misma aplicación a otras redes sociales

Uno de los problemas de realizar el estudio en Twitter es el tipo de usuario que utiliza Twitter ya que la media de edad es de 23 años por lo cual mucho del contenido de esta red no es fácil de validar.

Por problemas de escritura los usuarios utilizan todo tipo de caracteres para escribir sus tweets, tabuladores y una amplia gama de emoticonos que no ayudan al análisis de los tweets.

Una red como Facebook la media de edad de estos usuarios es mayor por lo que el contenido es más fácil de tratar y también en un gran porcentaje los tweets son de usuarios más fiables.

- Estudio sobre la previsión de votos sobre elecciones

Una gran idea sería que si estuviéramos en época de elecciones realizar un estudio de los tweets para saber una previsión de los votos, esta herramienta se vendería no solo a los partidos políticos sino a cientos de usuarios para conocer los que piensan los votantes.

- Buscar candidatos a suicidio

Es un tema horrible en nuestra sociedad pero las personas que están con trastornos debidos al estrés o otras causas a menudo buscan desahogó en Twitter.

Muchas pasan desapercibidas pero el índice de suicido, cada año entre 3.600 y 3.700 personas se suicidan en España es un dato a tener en cuenta pero estaría bien tener una herramienta que previniera el realizar esta práctica.

Con avisos a familiares, mandar ayuda al usuario en el caso de que estos tweets sean muy recurrentes.

- **Crítica a película o eventos**

Se que este tema se utiliza ya y que varias de las productoras de cines lo utilizan, bien cuando estrenan tráiler de sus películas, por ejemplo la película de Sonic, el aspecto del protagonista no gusto en las redes sociales y los productores de la película cambiaron al personaje para el disfrute del público. Pues extrapolarlo a otros eventos como festivales y programas de televisión.

- **Identificar casos de violencia de genero**

Cada vez más por desgracia hay más casos de violencia de género esto repercute en la sociedad de una manera negativa. Muchas de las "personas" que realizan estos actos tienen rasgos en común que pueden ser identificados.

- **Búsqueda de ofertas en artículos**

A menudo se buscan miles de ofertas para comprar artículos de moda o descuento, realizar una aplicación que busque todas las ofertas en Twitter sobre un articulo.

- **Aplicación Universidad de Jaén**

Una aplicación para la Universidad de Jaén en la que sus objetivos fueran los siguientes:

- **Información sobre la universidad**

Como llegar , facultades, mapa, zonas de interés, cafetería (menú de comidas), alquiler de pistas deportivas, reserva de libros y zonas de estudio.

- **Talleres**

La Universidad se puede enseñar de muchas formas, realizando talleres por estudiantes de doctorado, estos tienen gran experiencia por haber terminado la carrera y pueden aconsejar a los nuevos alumnos como afrontar la carrera, por medio de charlas.

- **Relación con los profesores**

La aplicación puede tener chats entre usuarios de una asignatura, no solo para hablar sobre temas relacionados con la asignatura, esto crea un mejor ambiente entre compañeros y ayuda al profesor para saber el nivel de la clase.

- **Test**

Si el profesor ve adecuado realizar test para saber si los conocimientos que esta impartiendo los están captando el alumnado.

- **Alertas**

Para que los alumnos estuvieran al tanto de todo lo que ocurre con sus asignaturas, un nuevo tema, comentarios de ejercicios, notas de una practica.

- **Modelo 3d de material sanitario**

Debido a la crisis mundial que está pasando a nivel mundial por el coronavirus y España

en especial, un modelo en 3d de mascarillas para que los hospitales y resto de material sanitario.

Dar la oportunidad a los hospitales y centros de salud de autoabastecerse en una crisis como esta en la que cada hospital o centro de salud tenga su propia impresora 3d para imprimir sus materiales, haría que una crisis de estas magnitudes se pueda llevar mejor.

- Sistema para teletrabajo para clases de la universidad

Realizar una plataforma propia de la universidad para llevar mejor el tema del teletrabajo con los alumnos de la universidad de Jaén. Esta aplicación en la cual el profesor pueda explicar sus clases y realizar ejercicios de forma que los usuarios tengan mayor facilidad para comprender la materia.

- Aplicación para la realización de ejercicios prácticos durante una cuarentena

Por desgracia hay algunos profesores que no tienen mucha habilidad con las nuevas tecnologías y sería bueno para el alumnado, con esta aplicación el profesor podría realizar ejercicios a mano y dar la opción al alumnado a ir viendo la resolución de estos.

Apéndice A

Diario de trabajo

La forma en la que se ha afrontado la planificación del Trabajo de Fin de Grado (TFG) ha sido buscando pequeñas tareas a lo largo de las semanas, bajo la supervisión del tutor del TFG, comenzando el trabajo con la primera entrevista que fue el 13 septiembre. La dificultad del trabajo se ha visto incrementada por ser un estudiante de la rama de Tecnologías de la Información, siendo la temática más cercana a la rama de Sistemas de Información.

Debido a esta razón se ha tenido antes que consultar el temario de la asignatura Procesamiento del lenguaje natural, así como el lenguaje de programación que iba a utilizar, ya que el tutor dio total libertad para elegir, aconsejando el uso de Python.

Se tuvo que realizar un estudio de los siguientes temas:

A.0.1. Búsqueda de información

- Apuntes de Procesamiento de Lenguaje Natural.
- Manuales o cursos del lenguaje Python.
- Estudio de la Api de Twitter y de los Wrappers en Python de la Api.

Habría que hacer un estudio para ver si es posible la realización de este proyecto, ya que era un trabajo teórico-experimental el cual no se tenía certeza de su resolución, se planteo continuar con este reto para seguir aprendiendo nuevos campos de la informática.

A.0.2. Planificación por etapas

Después de realizar un estudio de los puntos anteriormente descritos, habría que confeccionar un método para avanzar en el proyecto y recopilar este avance, pequeñas tareas ir dándole solución a los problemas que se planteaban, se cambiaba las tareas tras su realización

o por el transcurso de las tutorías semanales en las que el tutor, por su conocimiento más avanzado en estos temas, iba guiando.

- Día 30 septiembre 2019:

- Poner en marcha la Api en Python, búsqueda de usuarios.
- Estudio de límites de Tweepy.
- Búsqueda de un foco.
- Alerta por palabras clave.
- Búsqueda del tweet más reciente de un usuario.
- Distinguir palabras, hashtag y usuarios de un tweet.
- Escucha activa geográfica, de organizaciones.
- Búsqueda Usuarios relacionados sobre un tema.
- Búsqueda Hashtag.
- Poner en marcha la Api en Python, creación usuarios.
- Búsqueda Original (valores de entrada sin modificar).
- Búsqueda de Foco.

- Día 2 octubre 2019:

- Búsqueda Usuarios relacionados sobre un tema.
- Poner en marcha la Api en Python, creación usuarios.
- Búsqueda Original (valores de entrada sin modificar).
- Búsqueda de Foco.
- Búsqueda de trending topic respecto al valor original.
- Búsqueda de excepciones.
- Estructura de búsquedas como en multiagentes.
- Comparador de fechas de tweet para búsqueda de foco.

- Día 3 octubre 2019:
 - Interfaz de escritorio para la aplicación
 - Investigar Tkinter
- Día 6 octubre 2019:
 - Estimación de la repetición de cada tarea.
 - Estructura para guardar la información descargada.
- Día 10 octubre 2019:
 - Problema con la poca cantidad de tweets descargada.
 - Contenido repetido.
 - Búsqueda por recientes.
 - Búsqueda por popular.
 - Búsqueda mista.
 - Problema con la restricciones de la Api.
- Día 17 octubre 2019:
 - Detectar palabras clave en tweet.
 - Detectar localidades en tweet.
 - Detectar organizaciones en tweet.
 - Detectar usuarios mencionados en tweet.
 - Búsqueda para encontrar sinónimos o palabras relacionadas.
 - Extracción de las raíces de las palabras.
 - Boceto de la estructura nodo utilizada para guardar las partes de los tweets.
- Día 20 octubre 2019:
 - Estructurar las operaciones realizadas sobre los tweets
 - 1. Poner minúscula.

2. Quitar saltos de línea.
 3. Quitar stopwords.
 4. Quitar signos de puntuación.
 5. Eliminar enlaces en el texto.
 6. Identificar entidades.
 7. Identificar lugares.
- Día 25 octubre 2019:
 - Problema con ficheros vacíos
 - Límite de geonames, buscar alternativa
 - Día 8 noviembre 2019:
 - Pasos a la hora de procesar datos del usuario.
 1. Capturar valores.
 2. Quitar mayúsculas.
 3. Quitar stopwords.
 4. Valores combinados.
 5. Búsqueda Usuarios.
 6. Comenzar las búsquedas.
 7. Guardar.
 8. Análisis.
 - Día 22 noviembre 2019:
 - Tutor: cambiar interface de escritorio a web.
 - Estudiar: Flask, Django.
 - Estudiar librería Bokeh para gráficos.
 - Estudiar el uso de red neuronal para el proyecto.
 - Día 18 diciembre 2019:
 - Estructura para Usuarios.
 - Estructura para Hashtag.

- Estructura para Organizaciones.
- Estructura Foco.
- Estructura Wikipedia.
- Estructura valores entrada.
- Estructura localidades.
- Datos que se van a seleccionar de cada tweet.
- Día 19 diciembre 2019:
 - Gráfica Bokeh Nube puntos diccionarios.
 - Gráfica Bokeh Nube puntos clustering.
 - Gráfica localidades
 - Gráfica Organizaciones
 - Usuarios más nombrados
 - Tweets con más retweet
 - Tweets con más favoritos
- Día 11 enero 2020:
 - Estructura de datos para recolectar: Nombre, URL y texto
 - Estructura de datos para recolectar: localidades, URL, veces que han sido nombradas
 - Estructura de el análisis:
 1. Presentación, resumen de los valores de entrada.
 2. Análisis: tweet, numero tweets descargados, gráfica de usuarios más retweeteados, con más favoritos, análisis de organizaciones y localidades.
- Día 12 enero 2020:
 - Problema con el carácter de prueba para el csv, pruebas con multicarácter.
- Día 13 enero 2020:
 - Problema con la librería que reconoce las entidades, tarda mucho 40 min /1000 tweets

- Día 14 enero 2020:
 - Estudio de un sistema de puntuación de los tweets, para su evaluación, esto es debido a que la api de Twitter devuelve tweets por palabras clave.
 - Necesidad de palabras relacionadas con el valor entrante, los tesauros, posibilidad de uso de la Api de Wikipedia.
- Día 24 enero 2020:
 - Problema en la generación del agrupamiento, las palabras más usadas en los tweet son siempre determinantes y palabras stopwords, hay que realizar un stopwords propios más restrictivos que los de la librería.
- Día 3 febrero 2020:
 - Construir estructura periódica para las funciones que se repitan cada cierto tiempo.
 - Problemas con Bootstrap4 y Ajax
 - Buscar forma de encontrar el numero de cluster óptimo para el agrupamiento de los tweets.
- Día 8 febrero 2020:
 - Problema con las vistas y Flask, hay que mejorar el feedback con el usuario.
 - Problemas con los menú y Javascript.
- Día 15 febrero 2020:
 - Problema con los gráficos de Bokeh y los datos de Twitter, hay que realizar una limpieza de los datos que contienen caracteres raros y emoticonos tanto en el nombre de usuario, localización y texto.
- Día 23 febrero 2020:
 - Cambiar colores de las gráficas a colores más correctos.
 - Tutor: Desechar la idea de crear un pdf, generar un html con el contenido de la captura y el análisis.
- Día 10 marzo 2020:
 - Cambiar pestaña clustering por agrupamiento

- Añadir csv a la interfaz para que pueda copiarlos el usuario.
- Día 11 marzo 2020:
 - Problemas con la librería que encuentra las entidades, tiene un límite de 1.000.000 de caracteres, a partir de esta cifra hay que generar subconjuntos de análisis.
- Día 16 marzo 2020:
 - Experimento de 3 horas con palabra COVID"
 - Se sigue con la documentación
- Día 20 marzo 2020:
 - Problemas con la instalación en Mac os x ,después de terminar la ejecución Python se cierra.
 - Estudiando la forma de instalar el proyecto en Linux, Mac y Windows.
 - Seguimos con la documentación
- Día 26 marzo 2020:
 - Se sigue con la documentación
 - Completadas las instalaciones en los tres sistemas operativos.
 - Terminadas las excepciones de Wikipedia y de la clase clustering.
- Día 27 junio 2020:
 - Por consejo del tutor se cambió el menú

Apéndice B

Instalación

Se va a explicar una serie de pautas para instalar el TFG en un sistema operativo, se han elegido para instalarlo: Windows 10, Ubuntu 20.04 y Mac 10.13.6.

B.0.1. Ubuntu

1. Paso instalar python3.7

```
sudo apt-get install python3
```

Es la versión con la que se ha programado el TFG.

2. Instalar el modulo de paquetes pip

```
sudo apt-get install python3-pip
```

3. Despues instalar todos los paquetes que están en el apartado una vez instalado pip.

B.0.2. Windows

Instalar python 3.7 y descargarse el modulo pip para instalar paquetes.Despues instalar todos los paquetes que están en el apartado una vez instalado pip.

B.0.3. MacOSx

1. Instalar Homebrew

2. Instalar el siguiente linea:

```
$ ruby -e "$(curl -fsSL https://  
raw.githubusercontent.com/Homebrew/install/master/install)"
```

3. Instalar python 3.7

```
$ brew install python3
```

B.0.4. Una vez instalado Pip

1. instalar pandas

```
pip3 install pandas
```

Se utilizo para manipular los csv.

2. instalar spacy

```
pip3 install spacy
```

Se ha utilizado para filtrar las organizaciones, las localidades y las personas de renombre.

3. instalar la librería nltk

```
pip3 install nltk
```

Se ha utilizado para encontrar las palabras vacías.

4. Instalar el microframework flask

```
pip3 install flask
```

Es el servidor web que se ha utilizado.

5. instalar el modulo de flask que se utiliza para modificar la cache del programa. **pip3 install Flask-Caching**

6. Se necesitara el uso de la api de wikipedia para generar los diccionarios con las palabras de la definición de las palabras.

```
pip3 install wikipedia
```

7. Se necesita instalar tweepy que se utiliza para conectar con la api de twitter.

```
pip3 install tweepy
```

8. instalar el modulo view

```
pip3 install view
```


9. instalar pycorenlp
pip3 install pycorenlp
Se necesita para la conexión con el servidor de Stanford.
10. Se necesita también la librería con la que se va a medir el numero de cluster de KMeans.
pip3 install kneed
11. Se va a necesitar la gráfica para mostrar la elección del numero de cluster.
pip3 install matplotlib
12. Para la vectorización de los tweets se ha necesitado sklearn.
pip3 install sklearn
13. Instalar bokeh para la representación de los gráficos web.
pip3 install bokeh
14. instalar los paquetes con las localidades para que funcionen abrir un terminal y instalar:

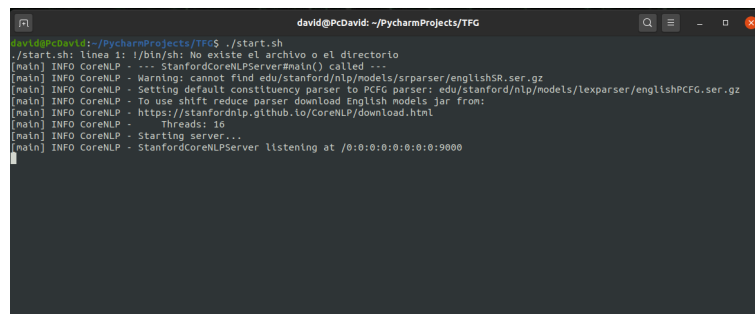
```
python -m spacy download es_core_news_md
```

```
python -m spacy download es_core_news_sm
```

B.0.5. Parseador Stanford

Para la hora de lanzar el proyecto, se tiene que tener en la carpeta del TFG el parser de Stanford, en el caso de tenerlo arrancar el servidor si es Linux o Mac con el script, start.sh antes dar permisos con el terminal. En el script viene configurado para un puerto si se cambia en las primeras lineas del main esta el enlace desde el proyecto al servidor para poner la ip correcta y el puerto.

Para el caso de windows se arrancar el servicio lanzándolo con java.

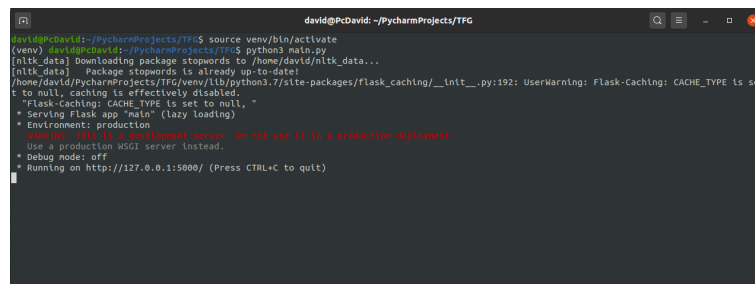


```
david@PcDavid: ~/PycharmProjects/TFG
david@PcDavid:~/PycharmProjects/TFG$ ./start.sh
./start.sh: línea 1: /bin/sh: No existe el archivo o el directorio
[main] INFO CoreNLP - StanfordCoreNLPServer#main() called --
[main] INFO CoreNLP - Warning: cannot find edu/stanford/nlp/models/srparser/englishSR.ser.gz
[main] INFO CoreNLP - Setting default constituency parser to PCFG parser: edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz
[main] INFO CoreNLP - To use shift reduce parser download English models jar from:
[main] INFO CoreNLP - https://stanfordnlp.github.io/CoreNLP/download.html
[main] INFO CoreNLP - Threads: 16
[main] INFO CoreNLP - Starting server...
[main] INFO CoreNLP - StanfordCoreNLPServer listening at /0:0:0:0:0:0:0:0:9000
```

Figura B.1: Parseador de Standford arrancado

B.0.6. Guia de uso

Una vez se ha lanzado el servidor que genera los analisis morfosintácticos, se carga el entorno virtual de python desde el terminal y justo después se lanza la aplicación de python.



```
david@PcDavid:~/PycharmProjects/TFG$ source venv/bin/activate
(venv) david@PcDavid:~/PycharmProjects/TFG$ python3 main.py
[nltk_data] Downloading package stopwords to /home/david/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
/home/david/PycharmProjects/TFG/venv/lib/python3.7/site-packages/flask_caching/__init__.py:192: UserWarning: Flask-Caching: CACHE_TYPE is set to null, caching is effectively disabled.
  "Flask-Caching: CACHE_TYPE is set to null, "
* Serving Flask app "main" (lazy loading)
* Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figura B.2: Iniciando el servidor flask

Se pincha en la dirección ip que se muestra en el terminal que sera la pagina web que muestre el index.html.

Se muestra una pequeña introducción del TFG y el menú de navegación, para realizar una captura se pinchara en la opción captura.

Se muestra un formulario con el tema general de busqueda, este valor sera el que se busque en Wikipedia, en el segundo formulario se introducen los valores para la subbusqueda y por ultimo el tiempo en segundos que es el tiempo de captura de tweets.

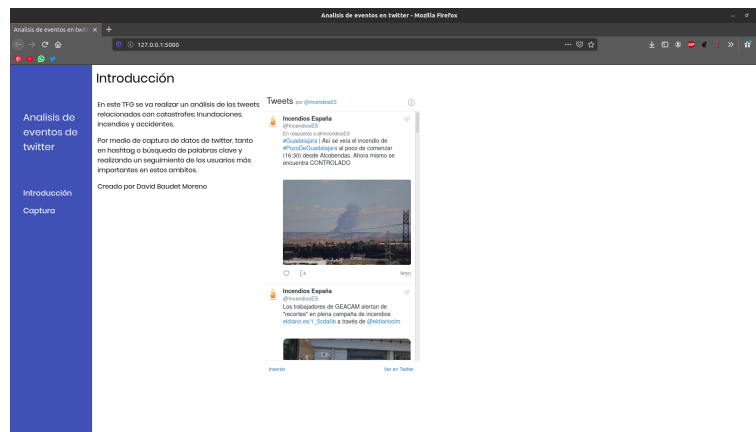


Figura B.3: Pagina inicio

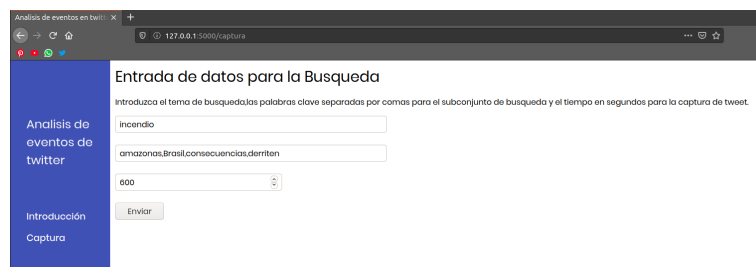


Figura B.4: Formulario de captura de valores de entrada

Se muestra un resumen de los valores y dos botones: Comenzar y abortar. El motivo por el uso de los botones es para darle al usuario la funcionalidad de cancelar el inicio de la ejecución por equivocación, el botón de comenzar inicia la rutina.

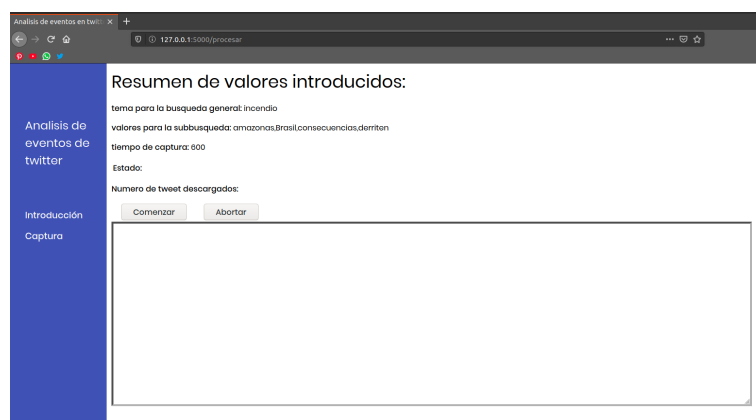


Figura B.5: Pagina de capturas de datos

La etiqueta de estado se ira cambiando a medida que se avance en la ejecución al final se muestran los resultados, se dan las opciones de descargar solo el archivo html y descargar el html junto los csv generados.

Análisis de la captura

Resumen de valores de entrada:

Numero de tweets: 2023

Valor de entrada para la búsqueda general: incendio

Valores para la subbúsqueda: amazonas,Brasil,consecuencias,derriban

Tiempo en segundos de la captura de tweets: 600

Medio de descargas del algoritmo de tiempo real: 1

Numero de medio de descargas del Búsqueda: 515

Numero medio de descargas de Búsqueda2: 991

Numero medio de descarga total: 502.3333333333333

Numero	Tweet
1	@SoyPetroSiva No explotó, se incendio la gasolina que saqueaban la comunidad olvidada de los gobiernos locales en magdalena por eso no hubo mas muerto.
2	Gobernador chavista de Falcon admitió que se produjo un incendio en la refinería de Cardón
3	El Titanic se hundió a causa de un incendio y no de un iceberg...
4	No dijo con la incendio.
5	¿Pien construya una nueva y mas avanzada caseta en la planta nuclear de #Nataraz para reemplazarla con la desde en un incidente reciente.
6	@marbeijn @NuPenaz Jajajaja tienes clarito lo que significa un conato de incendio.
7	Entre las incidencias de hoy podemos destacar el incendio de un vehículo cosechadora. El siniestro ha ocurrido en la A-3051, km 9. La duración de la intervención se prolonga mas de dos horas por las dificultades
8	No hay incendio como la pasión: no hay ringun mal como el odio. - Buta
9	Segun Victor Clark, gobernador del estado Falcon, incendio de refineria Cardón no afecta operaciones, ni causa heridos.
10	Incendio cobra vida de matrimonio y pequeño hijo en Rancagua
11	Se incendio una iglesia pero Alexis Zarela quedo intacta
12	10 Preguntas Frecuentes sobre Calentamiento Global SGK-PLANET #CambioClimatico #CalentamientoGlobal #EmergenciaClimatica #MedioAmbiente #Sostenibilidad #CrisisClimatica #AccionClimatica #Deserti
13	@LuboAhtuDF @SantaMarina Exploto? O solamente el combustible se incendio?

Figura B.6: Resultados de la captura

Apéndice C

Bibliografía

- [1] Jack Dorsey. Twitter. <https://es.wikipedia.org/wiki/Twitter>.
- [2] SINAI. Analisis de Sentimientos. http://timm.ujaen.es/wp-content/uploads/2014/03/analisis_de_sentimientos.pdf.
- [3] Machine Learning and Data Science | Bogotá. Workshops: Mi Primera Neurona + Analítica con Redes Sociales - Twitter. <https://www.meetup.com/es/Machine-Learning-Data-Science-Bogota/events/257945271/>.
- [4] Equipo TweetBinder. Tweet Binder. <https://www.tweetbinder.com/>.
- [5] Equipo Trendalia. Equipo Trendalia. <https://www.trendinalia.com/twitter-trending-topics/globales/globales-200702.html>.
- [6] Universidad de Jaén. Universidad de Jaén. <https://www.ujaen.es/>.
- [7] PLN.NET. Pagina de PLN.NET. <https://gplsi.dlsi.ua.es/pln/>.
- [8] SINAI. Pictogrammar, comunicación basada en pictogramas con conocimiento lingüístico. https://rua.ua.es/dspace/bitstream/10045/57773/1/PLN_57_27.pdf.
- [9] Flor Miriam Plaza del Arco, M. Teresa Martín Valdivia, Salud María Jiménez Zafra, M. Dolores Molina González, Eugenio Martínez Cámara. Pagina de PLN.NET. <https://gplsi.dlsi.ua.es/pln/>.
- [10] SINAI. Grupo de investigación SINAI. <http://timm.ujaen.es/grupo/sinai/>.
- [11] SINAI. SINAI en TASS 2018: Inserción de Conocimiento Emocional Externo a un Clasificador Lineal de Emociones. http://ceur-ws.org/Vol-2172/p15_sinai_tass2018.pdf.

- [12] JetBrains. Pycharm Pro. <https://www.jetbrains.com/es-es/pycharm/>.
- [13] JetBrains. JetBrains. <https://www.jetbrains.com/>.
- [14] Flask Team. Flask. <https://flask.palletsprojects.com/en/1.1.x/>.
- [15] Django Software Foundation. Django makes it easier to build better Web apps more quickly and with less code. <https://www.djangoproject.com/>.
- [16] Massimo Di Pierro . Web2py. <http://www.web2py.com/init/default/index>.
- [17] Twitter. Publish and analyze Tweets, optimize ads, and create unique customer experiences. <https://developer.twitter.com/en>.
- [18] Joshua Roesslein. Pagina de Tweepy. <http://docs.tweepy.org/en/latest/>.
- [19] Jeremy Low. Welcome to python-twitter's documentation! <https://python-twitter.readthedocs.io/en/latest/>.
- [20] WOEID. WOEID (Where On Earth IDentifier). <https://es.wikipedia.org/wiki/WOEID>.
- [21] Spacy. Part of-speech tagging. <https://spacy.io/usage/linguistic-features#pos-tagging/>.
- [22] NLTK Proyect. Text classification using k-means. <https://www.nltk.org/>.
- [23] stackoverflow. Natural Language Toolkit. <https://medium.com/@dennisndungu68/text-classification-using-k-means-33bea24e4a94/>.
- [24] Web scraping. Web scraping. https://es.wikipedia.org/wiki/Web_scraping.
- [25] UNESCO. Tesauro de la UNESCO. <https://skos.um.es/unescothes/?l=es>.
- [26] SPINES. Tesauro de la UNESCO. <https://bartoc.org/en/node/36>.
- [27] Universidad de Madrid. Tesauro Digital Complutense (TDC). <http://alfama.sim.ucm.es/tesauro/>.
- [28] Jonathan Goldsmith. Wikipedia API for Python). <https://pypi.org/project/wikipedia/>.
- [29] CoreNLP Server. Corenlp server pagina web.
- [30] Sarah Bird, Luke Canavan, Carolyn Hulsey, Mateusz Paprocki, Philipp Rudiger, Bryan Van de Ven. BOKEH. <https://docs.bokeh.org/en/latest/index.html>.

- [31] Mark Lutz. *Learning Python: Powerful Object-Oriented Programming*. International series of monographs on physics. O'Reilly Media; Edición: 5 (12 de junio de 2013).
- [32] David Beazley, Brian K. Jones. *Python Cookbook: Recipes for Mastering Python 3*. International series of monographs on physics. O'Reilly Media; Edición: 3 (10 de mayo de 2013).
- [33] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly).
- [34] Data Carpentry. Análisis y visualización de datos usando Python. <https://datacarpentry.org/python-ecology-lesson-es/aio.html/>.
- [35] Overleaf. Sections and chapters. https://www.overleaf.com/learn/latex/Sections_and_chapters/.
- [36] Overleaf. Bibliography management in LaTeX. https://www.overleaf.com/learn/latex/Bibliography_management_in_LaTeX/.
- [37] lineadecodigo.com. Parámetros POST con Flask. <http://lineadecodigo.com/python/parametros-post-flask/>.
- [38] stackoverflow. How to fire AJAX request Periodically? <https://stackoverflow.com/questions/5052543/how-to-fire-ajax-request-periodically>.
- [39] Adrián García Diéguez. TÉCNICAS DE CLUSTERING APLICADAS AL ANÁLISIS DE TRENDING TOPICS EN CONJUNTO DE TWEETS. https://e-archivo.uc3m.es/bitstream/handle/10016/22233/PFC_adrian_garcia_dieguez_2014.pdf?sequence=1&isAllowed=y.
- [40] José Carlos Sobrino Sande. Análisis de sentimientos en Twitter. <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81435/6/jsobrinostFM0618memoria.pdf/>.
- [41] Daniel Garnacho Martín. SISTEMA DE BÚSQUEDA Y ANÁLISIS BASADO EN TWITTER. https://repositorio.uam.es/bitstream/handle/10486/668799/Garnacho_Martin_Daniel_tfg.pdf?sequence=1&isAllowed=y/.
- [42] Pagina web SEPLN. Pagina de SePLN. <https://sepln2017.um.es/index.html>.
- [43] Ricardo Moya. Selección del número óptimo de Clusters. <https://jarroba.com/seleccion-del-numero-optimo-clusters/>.

-
- [44] stackoverflow. Tweepy. <https://stackoverflow.com/questions/50863789/in-flask-how-can-i-modify-html-with-code-in-python/>.
- [45] Lía González. TESAUIROS. <https://www.bibliopos.es/tesauiros/>.
- [46] arvkevi. kneed.