Trabajo de Fin de Grado

Análisis de sentimientos y reconocimiento de entidades en Twitter

Analysis of sentiments and entity recognition on Twitter

La Laguna, 30 de junio de 2018

Dª. **Rosa María Aguilar Chinea**, con N.I.F. 43.778.956-C Catedrática de Universidad adscrita al Departamento de Ingeniería de Sistemas y Automática de la Universidad de La Laguna, como tutora.

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

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

Que la presente memoria titulada:

"Análisis de sentimientos y reconocimiento de entidades en Twitter"

ha sido realizada bajo su dirección por D. **Jonás López Mesa**,

con N.I.F. 78.851.455-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de junio de 2018

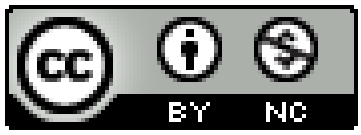
Agradecimientos

Quería agradecer a mi tutora Rosa María Aguilar Chinea por guiarme en el proceso de realizar este trabajo de fin de grado, por ayudarme a comprender un área de conocimiento nueva para mí y también por su predisposición a adaptar sus horarios de tutorías a mi horario de trabajo.

También quiero agradecer a Jesús Miguel Torres Jorge por su ayuda a la hora de desarrollar el proyecto y por su atención prestada ante todas mis dudas y problemas.

Por último, quiero agradecer el apoyo constante que he recibido por parte de amigos y familia en estos años de carrera y a Dios que tiene el control de todas las cosas.

Licencia



Resumen

En el presente proyecto, se pretende estudiar y comprender el procedimiento para desarrollar e implementar un aplicativo de análisis de sentimientos y reconocimiento de entidades utilizando Twitter. La idea principal del análisis de sentimientos se basa en algoritmos clasificadores que, habiendo sido entrenados previamente, son capaces de determinar qué tipo de sentimiento se desprende de un texto cualquiera, lo cual es muy útil para poder analizar cantidades masivas de comentarios acerca de un tema en particular. Por otro lado, el reconocimiento de entidades pretende descubrir si en un texto cualquiera se nombra alguna entidad de interés.

Se ha desarrollado un aplicativo multiplataforma, basado en Python y *QT for Python* que permite al usuario analizar *twits* de manera individual o en bloque, obteniéndolos desde Twitter.com o desde un fichero, para realizar un análisis de sentimientos así como poder reconocer entidades. Por otra parte, el aplicativo permite también entrenar los algoritmos clasificadores con diferentes parámetros para buscar la configuración más eficiente.

El resultado del presente proyecto será el ya mencionado aplicativo multiplataforma para análisis de sentimientos, reconocimiento de entidades y entrenamiento de clasificadores, además del estudio del comportamiento de dichos algoritmos y de los sentimientos y entidades que se desprenden para el tema propuesto del Carnaval de Tenerife.

Palabras clave: Análisis de sentimientos, Reconocimiento de Entidades, Twitter, Python, Algoritmos clasificadores, aprendizaje automático.

Abstract

In this project, the objective is to study and understand the procedure to develop and implement an application for the analysis of sentiments and recognition of entities using Twitter. The main idea of sentiment analysis is based on classification algorithms that, having been previously trained, are able to determine what kind of sentiment is derived from any text, which is very useful to be able to analyze massive amounts of comments about a particular topic. On the other hand, the recognition of entities is intended to discover whether an entity of interest is named in any text.

A multi-platform application has been developed, based on Python and *QT for Python* that allows the user to analyze twits individually or as a block, obtaining them from Twitter.com or from a file, to make an analysis of feelings and recognize entities. On the other hand, the application also allows to train the classification algorithms with different parameters to search for the most efficient configuration.

The result of the project will be the mentioned multiplatform application for the analysis of sentiments, recognition of entities and training of classifiers, as well as the study of the behavior of these algorithms and the sentiments and entities that emerge for the proposed theme of the Carnival of Tenerife.

Keywords: Sentiments analysis, Entity recognition, Twitter, Python, Classification algorithms, automatic learning.

Índice general

Índice de figuras

Índice de tablas

1. Antecedentes

1.1 Twitter

El microblogging, o el arte de transmitir mensajes cortos y nítidos a través de internet, se ha convertido en uno de los medios de comunicación más poderosos en los últimos tiempos. ¿La razón? La capacidad de transmitir en muy pocas palabras la opinión de las personas. En este sector, el líder indiscutible es Twitter.

Twitter, la famosa red social fundada en 2006 por Jack Dorsey y Noah Glass[1], cuenta con más de 320 millones de usuarios activos al mes y a pesar de que es una de las redes sociales con el crecimiento más lento, y que en los últimos meses incluso ha ido dibujando un decrecimiento paulatino de usuarios, es indispensable para la estrategia *social media marketing* debido principalmente a los siguientes aspectos:

- **El tiempo real:** Twitter es la plataforma, junto con Reddit, que muestra la actualidad antes que nadie. Esto se debe al hecho de que los mensajes que se escriben en Twitter son directos, enfocados exclusivamente a lo que se quiere decir, sin rodeos ni adornos. Esta característica lo hace propicio para ser un lugar donde se comparte todo lo que está pasando.
- **Medio de comunicación para las marcas:** Al ser un portal donde los mensajes deben ser claros y concisos, es perfecto para cualquier marca, empresa o anunciante que quiera publicitarse. Y no sólo de manera extra-oficial, sino que se ha convertido en el medio de comunicación oficial de la inmensa mayoría de las grandes y pequeñas marcas.
- **Pública:** Twitter es mayoritariamente pública, lo que permite a las marcas o a las empresas interesadas realizar una escucha social a través de ella.

En definitiva, Twitter es el lugar donde se puede escuchar la opinión "en vivo" de la gente sobre casi cualquier tema. Su naturaleza breve e instantánea permite publicar y consultar noticias de última hora sobre desastres naturales, elecciones políticas, eventos deportivos o el último concierto del artista de turno, lo que lo convierte en el medio perfecto para *medir la opinión.*

[1] Además de estas dos personas, participaron en su fundación Biz Stone y Evan Williams

1.2 Big Data

El Big Data, o los datos masivos, es el término que se utiliza para referirse a una cantidad tan sumamente grande de datos que las aplicaciones o algoritmos tradicionales que se suelen utilizar para procesar datos, no son suficientes para tratar esa información.

Fue en 1989, tres años antes del nacimiento de internet, cuando Erik Larson habló por primera vez acerca de Big Data [2]en el sentido que conocemos la expresión hoy en día. En este artículo Larson hablaba sobre el uso de los datos utilizados por los anunciantes para dirigirse a los clientes, lo que se convertiría en el principal uso que se le da hoy en día.

A partir de 1991 comienza la revolución de la red de redes, los avances en la tecnología hacen posible que sea más barato el almacenamiento digital que el papel, Google lanza su motor de búsqueda, en 2005 comienza la web 2.0 y aumenta aún más el volumen de datos, en 2009 se estima que una compañía promedio con 1.000 empleados almacena en torno a 200 terabytes de datos. En 2010, en una conferencia de Google, su presidente informa que la cantidad de datos que se generan cada dos días es mayor que la que se generó desde el comienzo de la civilización humana hasta 2003. En 2014 el uso de internet en los móviles supera a los ordenadores de escritorio por primera vez. Hoy en día, la mitad de la población humana se conecta diariamente a internet (más de 4.000 millones de personas) de las cuales más de 3.000 millones hacen uso de las redes sociales y el 90% desde dispositivos móviles.[3]

Si miramos concretamente el ámbito de este proyecto y nos centramos en Twitter, las últimas estadísticas indican que cada minuto se generan más de 480.000 tweets, lo cual significa que podemos contar con cerca de 5.000 millones de tweets nuevos por semana.

Este crecimiento desmesurado de los datos que se crean diariamente en todo internet hace que el desarrollo y mejora de nuevas técnicas enfocadas al tratamiento de grandes cantidades de información se haya convertido en una prioridad en las principales empresas de desarrollo de software. Las técnicas de Big Data no son un capricho o una rama de investigación alternativa, sino que es la única manera de abordar el problema de darle uso a las inmensas cantidades de datos que generamos.

[2] Concretamente dijo: Los guardianes del Big Data dicen que lo hacen en beneficio del consumidor. Pero los datos tienen una forma de ser utilizados para fines distintos a los originalmente previstos

[3] Estos datos cambian cada día, un lugar interesante para consultar esta información es internetWordStats.com

1.3 Análisis de Sentimientos

El análisis de sentimientos (o la minería de opinión) es el uso del procesamiento del lenguaje natural, análisis de texto y lingüística computacional para identificar y extraer información subjetiva de un recurso dado, en nuestro caso, de cualquier texto. En este sentido, el análisis de sentimientos se basa única y exclusivamente en detectar y extraer esa información de un caso concreto, pero si lo vemos desde el punto de vista de la minería de datos, el análisis de sentimientos se basa en clasificar masivamente documentos (tweets) de manera automática en función de la connotación que tenga ese documento. De esta manera, el análisis de sentimiento no realiza un análisis lingüístico del documento, sino que, generalmente, se basan en relaciones estadísticas y de asociación. Dicho de otro modo, el análisis de sentimientos en la minería de datos no trata de comprender lo que dice un texto para determinar si es positivo o negativo, sino que utiliza lo que sabe acerca de otros textos para determinar el sentimiento de otro texto en concreto.

El análisis de sentimientos tenía un uso bastante limitado antes de la llegada de redes sociales. [4]Los clasificadores que existían se centraban mayoritariamente en clasificar correos electrónicos como SPAM, pero no existía un uso dedicado exclusivamente al comercio y al marketing. Sin embargo, con la proliferación de las críticas, calificaciones, recomendaciones y opiniones de todo tipo de redes sociales y páginas webs, la opinión de las personas se ha convertido en una divisa para aquellas empresas que buscan venderse.

Es por ello que el análisis de sentimientos tiene ahora un alcance y un uso mucho más importante que nunca y aunque aún queda mucho por desarrollar y, en concreto, el problema de analizar sentimientos dado un texto es bastante complejo incluso para un ser humano, estas técnicas cada vez más se convierten en el punto de mira del desarrollo e investigación.

1.4 Reconocimiento de entidades

El Reconocimiento de Entidades Nombradas (o NER por sus siglas en inglés) son técnicas que consisten en extraer información de un documento y clasificarla en categorías predefinidas. Estas categorías suelen ser personas, organizaciones, lugares, expresiones, etc.

El uso de NER se encuentra en un estado mucho más avanzado que el análisis de sentimientos. En el caso del inglés, los sistemas de reconocimiento de entidades tienen un rendimiento muy cercano al del ser humano. El último y mejor sistema desarrollado, y presentado en el MUC-7 (Una conferencia de comprensión del mensaje financiada por DARPA) obtuvo una puntuación de precisión del 93.39%

[4] Es interesante leer el estudio The Evolution of Sentiment Analysis por Mika V. Mäntylä, concretamente el apartado 3.1 Una breve historia del análisis de sentimientos, donde se dan tres breves ejemplos del uso antes de la llegada de internet.

[5]estando la de los humanos entre el 96.95% y 97.60%

El reconocimiento de entidades generalmente se divide en dos partes bien diferenciadas: Detectar nombres y clasificarlos según el tipo de entidad:

- Detectar nombres: En este caso, se trata de un problema de segmentación. Los nombres son una secuencia de tokens que ni se solapan ni se anidan, de tal manera que Real Madrid debería ser considerado como un nombre único, a pesar de que Madrid por si sólo es también un nombre.
- Clasificarlos por entidad: En la mayoría de los casos, el problema con clasificar los nombres es que estos nombres dependen de un contexto que viene con el texto, y que, al extraerlo, se puede diluir el significado original. Por lo tanto, en muchas ocasiones, el contexto debe ser explicado.

[5] Para más información acercad de estos resultados, se puede consultar el estado del arte del MUC-7

2. Estado del arte

2.1 Análisis de Sentimientos

En la actualidad, el análisis de sentimientos se puede clasificar en cuatro categorías principales:

- Localización de palabras clave: Este método trata de clasificar el texto según palabras clave que no tengan ambigüedad (como contento, asustado, alegre, feliz, etc.) Mediante estas palabras determina el sentimiento del texto.
- Afinidad léxica: En este caso, se hace lo mismo que con la localización de palabras clave, pero además se añade a palabras arbitrarias una afinidad con emociones particulares. De manera práctica, podríamos suponer que aumenta la precisión del sentimiento en concreto.
- Métodos estadísticos: Este método utiliza la capacidad de ciertas ramas de la estadística y la computación como el aprendizaje automático (que permite a un ordenador aprender), la indexación semántica latente (que se aprovecha de que las palabras utilizadas en el mismo contexto suelen tener significados similares para identificar patrones entre palabras), etc.
- Técnicas a nivel de concepto: Por último, esta técnica es la más compleja ya que no sólo se queda con lo que significan las palabras, sino que intenta extraer el sentido dentro del contexto del resto de palabras, haciendo la información mucho más fiable. Parte de la potencia propia de estas técnicas es que son capaces de extraer recursos irónicos o similares.

Para poder implementar un análisis de sentimiento de forma rápida se han creado herramientas de software de código abierto que despliegan técnicas de aprendizaje de máquinas, técnicas estadísticas y técnicas de procesamiento de lenguaje natural para hacer el aprendizaje de manera automática en grandes colecciones de textos. Las técnicas que vamos a utilizar en este proyecto tienen su base en estas tres componentes, pero para abordar el problema del aprender de manera automática a partir de grandes cantidades de información es necesario entender las técnicas conocidas como *machine learning*.

2.1.1 El aprendizaje de máquina

También conocido como el aprendizaje automático, es un concepto que se refiere a la capacidad que tiene un programa u ordenador de aprender mediante procesos de inducción del conocimiento.

Tiene un amplio abanico de aplicaciones en motores de búsqueda, diagnósticos médicos, detección de fraude, análisis de mercados de valores, clasificación de secuencias de ADN, y también, como es nuestro caso, reconocimiento del habla y del lenguaje escrito.

Dentro de los sistemas de aprendizaje automático, hay algunos que se enfocan en eliminar toda la participación del ser humano en toma de decisiones, es decir, que intentan eliminar el conocimiento de una persona a la hora de evaluar un problema. Un ejemplo podría ser el análisis de secuencias de ADN. Sin embargo, con el tiempo el desarrollo ha demostrado que la intuición humana y el conocimiento de las personas no puede ser reemplazado por ningún programa. Por ejemplo; un programa de diagnóstico médico, que recibiendo una serie de síntomas devuelve un diagnóstico de posible enfermedad, no puede reemplazar a la opinión experta de un médico, ya que para determinar un diagnóstico no sólo es necesario conocer los síntomas, sino que existen otros contextos que podrían determinar la enfermedad real.

Dentro del aprendizaje de máquina se pueden distinguir diferentes modelos para las diferentes tareas que se quieren resolver: Modelos geométricos, Modelos lógicos y Modelos probabilísticos. Además, dentro de estos modelos, se pueden dividir los tipos de algoritmos en función de la información que revelen: Aprendizaje supervisado, aprendizaje no supervisado, aprendizaje por refuerzo, aprendizaje semi-supervisado, etc. En nuestro caso, nos centraremos en el aprendizaje supervisado:

Aprendizaje supervisado

Un algoritmo que se basa en el aprendizaje supervisado es aquel que deduce una función a raíz de unos datos de entrenamiento previamente utilizados. Estos datos de entrenamiento consisten en un listado de pares de datos que corresponden a:

- Datos de entrada: En nuestro caso sería el texto de un tweet.
- Resultado deseado: En nuestro caso el sentimiento que provoca dicho tweet (positivo, negativo o neutro)

Lo que devuelve el algoritmo es una etiqueta de la clase que se ha establecido. En resumen, un algoritmo de aprendizaje supervisado primero observa cómo se categorizan una serie de datos, y una vez haya aprendido lo suficiente, está listo para determinar y clasificar una entrada nueva distinta a las que se le ha dado para entrenar.

En el caso concreto de reconocer texto, una persona que se disponga a hacer uso

de un algoritmo de aprendizaje supervisado debe atender a lo siguiente antes de realizar el entrenamiento:

- Determinar el ámbito del estudio: Es muy importante, antes de realizar el entrenamiento, establecer cuál va a ser el ámbito o el contexto de los datos que se van a analizar. Si mi objetivo es buscar información acerca de los sentimientos de un festival de música que se va a celebrar en el próximo mes tendría que recopilar unos datos de entrenamiento que tengan relación, como mínimo, con conciertos, festivales, música, artistas, cantantes, etc. En este caso hay dos variantes:
    - El contexto de entrenamiento es general: En este caso, tenemos muchas posibilidades de que el entrenamiento no sea bueno. El contexto general impide que el algoritmo pueda entrenarse con el vocabulario y matices propios de un contexto específico.
    - El contexto de entrenamiento es distinto: Es el peor caso posible, ya que no sólo se entrenará con un vocabulario y expresiones distintas sino que seguramente existirán expresiones del lenguaje que utilizando las mismas palabras quieran decir todo lo contrario.
- Establecer un número adecuado de datos de entrenamiento: Los datos utilizados en el entrenamiento de un algoritmo de aprendizaje supervisado se dividen en dos tipos:
    - Datos de entrenamiento: Datos donde se especifica al algoritmo cómo funciona el análisis. Dicho de otra manera, con estos datos el algoritmo aprende. Por ejemplo; el par de datos: *[“No puedo creerme que haya salido Trump. La que nos espera”; Negativo*] Pasado como dato de entrenamiento, está enseñándole al algoritmo que ese texto, con esas palabras, da un mensaje negativo.
    - Datos de prueba: Datos que se utilizan para determinar si el algoritmo ha realizado un entrenamiento preciso. Estos datos están convenientemente etiquetados, de la misma manera que los datos de entrenamiento, sin embargo, el algoritmo no los utiliza para entrenar, sino que los usa para averiguar si ha entrenado bien. Utilizando estos datos de prueba, a la par que los datos de entrenamiento, se puede ir ajustando la precisión del algoritmo. La cantidad de datos de cada tipo debe ser aproximadamente un 80% de datos de entrenamiento y un 20% de datos de prueba del total.

2.2 Reconocimiento de entidades

A pesar de que el reconocimiento de entidades podría parecer un problema más sencillo que el análisis de sentimientos, las investigaciones recientes indican que incluso los sistemas NER más potentes son débiles.

Esto es debido a que un sistema desarrollado para un dominio específico suele comportarse mal en un dominio distinto y un sistema desarrollado de manera generalista no suele funcionar correctamente en casi ningún contexto.

En la actualidad, a pesar de que los resultados de precisión son bastante buenos, el problema de reconocer entidades dista bastante de estar solucionado. La investigación se está centrando en reducir el trabajo de anotación (es decir, de hacer aprender al algoritmo) por medio de aprendizaje semi-supervisado para que sea mucho menos costoso cambiar el dominio del algoritmo. Además, se está investigando técnicas para hacer más fuerte el rendimiento de un algoritmo en varios dominios distintos y por último se está tratado de lograr los mismos resultados que se obtienen ahora con tipos de entidades no tan generales.

3. Objetivos

3.1 Objetivo general del proyecto

Comprender y evaluar el funcionamiento de las técnicas de análisis de sentimiento y reconocimiento de entidades, así como desarrollar un aplicativo que implemente dichas técnicas y muestre los resultados obtenidos. Concretamente se pretende desarrollar un aplicativo utilizando *Qt for Python* mediante el cual se pueda obtener datos de Twitter.com o desde un archivo para entrenar cuatro tipos distintos de algoritmos y una vez entrenados se pueda obtener información acerca de las entidades y sentimientos que se desprenden de cualquier tweet o bloque de tweets que se desee analizar.

3.2 Objetivos específicos

Con el desarrollo de este proyecto se pretenden conseguir los siguientes objetivos:

- Realizar un análisis de sentimientos para la temática del Carnaval de Tenerife.[6]
- Crear un aplicativo para ordenador en Python implementando técnicas de análisis de sentimientos y reconocimiento de entidades.
- Estudiar el funcionamiento y los resultados obtenidos de los entrenamientos y análisis realizados.
- Realizar una síntesis del funcionamiento de los cuatro algoritmos implementados.
- Creación de documentación que será utilizada para entender el funcionamiento del aplicativo, enfocado a un posible programador que trabaje sobre futuras versiones de la misma, o que quiera realizar modificaciones.[7]

[6] Este objetivo, como se explica en las conclusiones, fue cambiado, debido a la falta de tweets disponibles para realizar un estudio correcto.

[7] La documentación se encuentra en el anexo I

4. Fases y desarrollo del proyecto

A continuación, se muestra una tabla con la planificación del proyecto. Inicialmente el proyecto, tal como estaba concebido en el modelo del proyecto entregado en el campus virtual, constaba de los puntos del 1 al 4, pero conforme se avanzaba en la realización del mismo, se fueron añadiendo más actividades, resultando en la tabla siguiente:

Semana	Actividades	Puntos
1. Investigación		
19-25 febrero	Comprensión y definición del proyecto	
26-4 marzo	Revisión de la bibliografía e investigación autónoma	
2. Obtención de datos		3
5-18 marzo	Desarrollo del script inicial	
19-25 marzo	Obtención de datos mediante métodos del script y almacenamiento en base de datos	
3. Preparación de los datos		3
26-8 abril	Etiquetado, limpieza y tokenización de los datos	
9-22 abril	Envío de datos obtenidos para etiquetado previo al entrenamiento	
4. Detección de sentimientos con el algoritmo VADER		2
23-29 abril	implementación del algoritmo VADER para detectar sentimientos	
5. Detección de sentimientos con el algoritmo entrenado para el concepto		2
1-13 mayo	Implementación de algoritmo Naive Bayes	
14-20 mayo	Implementación de NER	
6. Otros clasificadores y comparativa de clasificadores		
21-27 mayo	Implementación de otros clasificadores	

28-3 junio	Desarrollo completo del script de comparativa de clasificadores implementados
7. Creación de interfaz gráfica	
4-10 junio	Limpieza de código e implementación con Qt
11-24 junio	Terminado aplicativo
8. Documentación	
25-1 julio	Realización de memoria

Tabla 1: Planificación para el desarrollo del proyecto

4.1 Metodología utilizada

La metodología que más concuerda a la llevada a cabo en el desarrollo del proyecto y a este tipo de proyectos es la metodología XP.

La programación extrema (XP) es un tipo de metodología ágil de ingeniería de software creada por Kent Beck [8]y que destaca, al igual que muchas metodologías ágiles en que pone más énfasis en la adaptabilidad que en la previsibilidad. Por ello se considera que los cambios de requisitos durante un proyecto son aspectos naturales del mismo, inevitables e incluso algo que se debe incentivar dentro del desarrollo. Adaptarse a los cambios de requisitos es una situación más realista que tratar de preverlos todos antes del comienzo del mismo. Sus principales características son:

- Pruebas unitarias continuas.
- Diseño simple.
- Desarrollo iterativo e incremental.
- Objetivos de desarrollo a corto plazo. Entregas pequeñas.

En el caso de este proyecto, esta metodología no se seguía de manera proporcional en periodos de tiempo, sino que iba variando en función de la dificultad que suponía cada tarea en concreto y también en función de los cambios que se realizaban durante el proyecto.

1. Cada periodo de tiempo proponía objetivos a cumplir durante ese periodo. Llevaba el conteo de las tareas realizadas y las tareas que quedaban por realizar mediante la utilización de un blog. De esta manera, no sólo tenía registrado qué tareas tenía que hacer y qué tareas había hecho, sino que redactaba todos los problemas que me iba encontrando para ser capaz de resolverlos en futuras ocasiones.

[8] Kent Beck también fue el creador del desarrollo guiado por pruebas así como de múltiples publicaciones enfocadas al desarrollo ágil.

2. Durante cada periodo de tiempo añadía objetivos a conseguir. Si conseguía terminar los objetivos antes del periodo previsto tenía dos opciones, continuar el desarrollo añadiendo más objetivos o investigar en profundidad la temática del proyecto para futuros objetivos.
3. Cada vez que comenzaba una tarea documentaba todos los problemas que me iba encontrando. Cuando acababa la tarea realizaba las pruebas correspondientes para determinar si debía continuar con otra tarea o debía reparar el erro que tenía en esa tarea.
4. Cuando terminaba un bloque de tareas concreto evaluaba el estado global del proyecto. Realizaba pequeñas limpiezas de código, así como comentarios que ayudaran a entender el código para futuras revisiones.
5. También en periodos de tiempo variable tenía reuniones con la directora del proyecto o establecía contacto por medio del correo electrónico para informarle del estado actual del proyecto, así como para verificar si estaba realizando las tareas de manera correcta. De la misma manera preguntaba casos concretos de hacia qué lugar se dirigía el proyecto. Esto me daba una buena base para saber en qué tareas debía poner más esfuerzo y priorizar.
6. Como parte del desarrollo ágil, trataba de esquematizar, siempre que me fuera posible la codificación que iba a realizar. Gracias a la documentación ofrecida por la tutora, en muchos de los casos este paso resultó sencillo debido a que era intuitivo trasladar las ideas al código.

4.2 Entornos de trabajo

En este apartado se nombrarán las herramientas utilizadas en el desarrollo del proyecto. En la mayoría de los casos utilicé las últimas versiones de todas las librerías y programas. Por otra parte, por recomendación de la tutora, utilicé herramientas que no había utilizado nunca antes y que resultaron de ayuda y gran utilidad.

- **Anaconda Navigator:** Se trata de una *versión*[9] de Python que implementa varias herramientas muy útiles para la utilización de Python en la investigación. Contiene un instalador que utiliza los repositorios de Conda para instalar cualquier tipo de librería para Python que pueda existir. En especial fue útil porque me sirvió para instalar de forma sencilla todos los paquetes para el tratamiento de grandes volúmenes de datos.
- **Jupyter Notebook:** Es una aplicación web de código abierto que permite crear y compartir documentos que contienen *código en vivo*. Esto quiere decir que la ejecución de un bloque de código se visualiza en el mismo editor justo debajo del bloque en concreto. Permite añadir bloques tanto de

[9] Conda podría identificarse como una gestión de paquetes, dependencias y entornos para varios lenguajes de programación, entre ellos Python, Anaconda Navigator, utiliza Conda, pero de manera práctica lo que nos interesaba de Conda es su implementación de Python y la facilidad de integrar librerías y entornos de trabajo.

código como markdown, lo cual es muy útil para documentar cada bloque de código de una manera más clara.

- **Mongo DB:** Mongo DB es una base de datos de datos que almacena la información en documentos flexibles similares a JSON, lo que significa que los campos pueden variar de un documento a otro y la estructura de datos puede cambiar con el tiempo.
- **Github:** Se trata de una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones de Git. Como el trabajo fue realizado en dos plataformas diferentes (macOS y Windows) y dos puestos distintos de trabajo, GitHub fue de mucha utilidad para poder hacer un buen desarrollo en plataformas distintas.
- **Librerías importantes:** SKLearn, Pandas, NLTK, QT for Python.[10]

4.3 Fundamentos tecnológicos

4.3.1 Bases de datos NoSQL

Las bases de datos NoSQL[11] se caracterizan por diferir del modelo clásico del Sistema de Gestión de Bases de datos relacionales. En concreto, el principal aspecto es que no usan SQL como lenguaje principal de consultas. Pero además hay otros aspectos que las hacen muy distintas de las bases de datos clásicas:

- Los datos almacenados no requieren estructuras fijas como tablas.
- No soportan operaciones de unión de tablas.
- No pueden garantizar la atomicidad, consistencia, aislamiento y durabilidad de la información.

La causa principal del nacimiento de este tipo de base de datos fue el gran crecimiento que tuvieron las principales potencias de internet como Google, Amazon, Facebook, etc. La gran cantidad de información que mueven estas bases de datos y la imperiosa necesidad de que todo fuese en tiempo real hacían imposible que se siguieran utilizando las estructuras básicas de bases de datos, ya que estas empleaban demasiado tiempo en tareas que debían durar muy poco.

Dentro de esta tecnología existen ventajas y desventajas:

Ventajas:

1. Son muy importantes para la escalabilidad horizontal (es decir, la posibilidad de aumentar el rendimiento de la base de datos simplemente añadiendo más campos)

[10] Serán explicadas en capítulos poseriores.

[11] Interesante y breve entrada en la Wikipedia acerca de los inicios del término y la utilización de bases de datos NoSql

2. No se generan cuellos de botella.
3. Escalamiento sencillo.
4. Pueden manejar enormes cantidades de datos.

Desventajas:

1. No tienen un soporte igual de competente que el de las grandes empresas de bases de datos.
2. Al ser de código abierto pueden presentar fallos o imposibilidades de uso.
3. No existen estándares, lo cual genera problemas de compatibilidad.

En el caso que nos ocupa, MongoDB es una base de datos creada por 10gen[12] del tipo orientada a documentos, de esquema libre, es decir, que cada entrada puede tner un esquema de datos diferente que nada tenga que ver con el resto de registros almacenados. Es bastante rápido a la hora de ejecutar sus operaciones ya que está en datos binarios. Para el almacenamiento de la información utiliza un sistema propio de documento conocido por el nombre BSON, que es una evolución de JSON con la particularidad de que añade dos campos: un campo para fecha *Date* y un campo para poder almacenar un array de bytes.

4.3.2 Algoritmos clasificadores

A modo de introducción y de realizar una vista general, en este punto se detalla brevemente una explicación acerca de los principales algoritmos clasificadores. En puntos posteriores veremos más información acerca de cada uno de ellos.

Como vimos anteriormente, los algoritmos clasificadores se pueden dividir en varios grupos, entre los cuales destaca los algoritmos de aprendizaje supervisado y los algoritmos de aprendizaje no supervisado.

En nuestro caso, los algoritmos son de aprendizaje supervisado, entre los cuales, los tipos de algoritmos más importantes son los siguientes:

- **Árboles de decisión:** Un árbol de decisión es una técnica que utiliza un gráfico en forma de árbol de decisiones y sus posibles consecuencias, entre las cuales se encuentran los resultados de eventos fortuitos, los costes de los recursos y la utilidad de las decisiones. Un árbol de decisión consta de los siguientes elementos:
    - **Hojas:** representan las etiquetas de clase
    - **Ramas:** representan las conjunciones de características que conducen a esas etiquetas de clases.

  En el campo de la minería de datos, un árbol de decisión describe datos,

[12] Aunque en su momento la empresa se llamaba 10gen, en 2013 cambiaron el nombre por el de su producto, MongoDB.

pero no las decisiones. Podemos encontrar diversos tipos de algoritmos específicos de árbol de decisiones como: ID3, C4.5, ACR, CHAID, CART.

- **Naïve Bayes:** (o clasificador Bayesiano ingenuo) Se trata de un tipo de clasificador probabilístico muy sencillo basados en la aplicación del teorema de Bayes. Para explicarlo brevemente, un clasificador Bayesiano asume que la presencia o ausencia de una característica en particular no se relaciona con la existencia o no existencia de otra característica cualquiera. Por ejemplo, un mueble puede ser considerado una silla si tiene cuatro patas, respaldar y es de madera; Un clasificador Bayesiano considera que cada una de estas características citadas contribuye a la probabilidad de manera independiente para que ese mueble sea una silla; no importa si las demás se dan o no.
- **SVC:** Por sus siglas en inglés (*Support Vector Classificator*) es un algoritmo de clasificación binario. El funcionamiento se basa en que el conjunto de datos se divide en dos categorías y cada dato se marca como perteneciente a una u otra categoría, a continuación se separan las categorías en dos espacios lo más amplios posibles mediante un hiperplano definido como el vector entre los dos puntos, de las dos clases, más cercanos el cual es llamado el vector de soporte. Cuando llegan nuevas muestras, el vector de soporte es el que determina a qué clase pertenece cada dato.
- **K-Vecinos más próximos:** Se trata de un método de clasificación (supervisada) que estima la función de densidad, o directamente la probabilidad, para cada elemento según los distintos tipos de clases. Se puede clasificar este algoritmo como un algoritmo de aprendizaje vago, ya que demora la generalización del aprendizaje hasta que se hace una pregunta al sistema, es decir, hasta que se le da un dato a clasificar.

4.3.3 NER

El Reconocimiento de Entidades Nombradas es parte fundamental del desarrollo de este proyecto. En nuestro caso vamos a utilizar el NER que ha desarrollado la universidad de Stanford.[13]

Esta implementación incluye extractores de funciones bien diseñados para el Reconocimiento de Entidades además de tener a disposición muchas opciones para definir extractores de características.

Todas las opciones son en inglés (salvo una que está en chino) y permiten identificar 3, 4 y 7 entidades, entre las cuales se incluyen: Lugares, organizaciones, fechas, dinero, personas, porcentajes y tiempo u hora.[14]

[13] Realmente se diseñó para Java, pero tiene adaptaciones para Python. Para obtener más información acerca del algoritmo desarrollado por Stanford, es recomendable consultar su web.

[14] En el caso del algoritmo que utilizamos en este proyecto, las entidades son: Lugares, Organizaciones y Personas.

4.3.4 API de Twitter

La Interfaz de programación de aplicaciones (API) proporcionada por twitter contiene el conjunto de datos, funciones y procedimientos necesarios para poder extraer la información de twitter y usarla dentro de nuestras aplicaciones.

Aunque no existe ninguna problemática con este punto y su utilización es trivial, es importante entender cómo funciona para implementaciones por parte de otros usuarios. Este tema concretamente lo abordaremos en puntos posteriores.

4.3.5 QT for Python

QT es un framework multiplataforma orientado a objetos que se usa principalmente para desarrollar software que utilicen una interfaz gráfica de usuario. Este framework utiliza el lenguaje de programación C++ de manera nativa, sin embargo, por medio de la utilización de *bindings* (adaptaciones de bibliotecas para ser usadas en un lenguaje de programación distinto de aquel en el que ha sido escrita) Qt se puede utilizar en otros lenguajes, como, por ejemplo, Python.

Para este propósito hay dos *bindings* que lideran el uso de Python con Qt[15]. *PyQt* y *Qt for Python*. En nuestro caso, utilizamos *Qt for Python*, que es el antiguo *PySide* (también llamado PySide2) por recomendación del cotutor del proyecto.

4.4 Descripción de procesos

Dentro de los procesos que se realizan en el aplicativo, existen partes claramente diferenciadas del resto, al margen de NER y los propios algoritmos, que deben ser descritas para una correcta y completa comprensión del funcionamiento del programa. En esta sección vamos a describir en profundidad cada uno de esos procesos.

4.4.1 Extracción de los tweets de twitter.com

El procedimiento de extracción de los tweets de Twitter.com se puede realizar de dos maneras distintas, y aunque en el apartado de Elementos del prototipo y funcionamiento veremos por encima los dos tipos, en esta ocasión vamos a centrarnos en el funcionamiento de uno de ellos, concretamente: *REST API*.

REST API Search endpoint permite realizar búsquedas históricas dentro de twitter para extraer los tweets que necesitemos. Dentro del código utilizado resaltan tres puntos importantes:

1. **Consulta a realizar:** El primer paso que tenemos que dar cuando queremos hacer una consulta a Twitter es indicarle qué estamos buscando, esto se realiza mediante una petición cuyo primer parámetro es una cadena de texto; esta cadena de texto es una consulta de la información que

[15] También existen otras alternativas como PyGTK o wxPython

queremos encontrar. Twitter permite insertar una serie de operadores y filtros para hacer la búsqueda más precisa. Utilizamos estos operadores porque queremos ahorrarle trabajo al proceso de limpieza de tweets. Entre estos operadores podemos encontrar:

a. **Operadores lógicos:** Los operadores lógicos permiten relacionar cadenas de búsquedas. Por ejemplo, escribiendo '*Carnaval AND Tenerife*' buscará tweets que contengan la palabra Carnaval y Tenerife. Si por otro lado escribimos '*Carnaval OR Tenerife*' encontrará tweets que puedan tener la palabra Carnaval, tweets que puedan tener la palabra Tenerife o Tweets que puedan tener ambas palabras.

b. **Escribir la cadena entre comillas dobles:** Si queremos que una cadena de caracteres se busque de manera literal sólo tenemos que poner dicha cadena entre comillas dobles. Si buscamos '*Carnaval Tenerife*' simplemente, la búsqueda puede tener esos conceptos o no, incluso buscará usuarios, hilos, etc. Sin embargo, si escribimos "*Carnaval Tenerife*" se buscarán tweets que contengan exactamente esa cadena, con mayúsculas y con un espacio entre medio.

c. **Símbolo menos:** Este símbolo es utilizado para eliminar la siguiente cláusula del resultado. Si escribimos algo como esto: '*Carnaval Tenerife -From:Jolome7*' Buscará todos los tweets que coincidan con la búsqueda pero que no los haya escrito el usuario *Jolome7*

d. **Filtros:** Los filtros permiten escoger características concretas dentro de las búsquedas. De esta manera podemos tener los siguientes filtros:

   i. **Retweets:** Filtra por tweets que sean Retweets.

   ii. **Replies:** Filtra por tweets que sean respuestas a otros tweets.

   iii. **Links, Images, Videos:** Filtra por tweets que contengan o enlaces, imágenes o vídeos.

2. Parámetros de la consulta: Esa consulta de texto se añade a un vector de consulta que es el que luego se envía a Twitter. Dentro de ese vector de consulta se encuentran los siguientes parámetros a tener en cuenta:

   a. **Q:** La consulta en una cadena de Texto. Como máximo puede ser de 500 caracteres.

   b. **Count:** Indica el número de tweets que serán solicitados en cada llamada, el máximo es 100 por llamada. Por defecto es 15.

   c. **Lang:** Se trata de una cadena de caracteres en la que se establece,

mediante los códigos de idiomas[16], el idioma o los idiomas en los cuales se va a buscar los tweets.

d. **Result_type:** Indica el tipo de resultado que se va a obtener. Puede ser de tres tipos: *recent* (devuelve los más recientes), *popular* (devuelve los más populares) o *mixed* (devuelve una mezcla de los más recientes y más populares).

e. **Max_id:** devuelve los resultados con un ID menor que (es decir, más antiguo que) el especificado en este parámetro. Imprescindible para realizar la paginación

3. **Paginación:** Como acabamos de ver (y veremos brevemente más adelante) Twitter permite un máximo de 100 tweets por llamada. Es por ello que se realiza un procedimiento de paginación para poder *sacar* de Twitter más de 100 tweets. El procedimiento consiste en que cada vez que realizamos una iteración en el bucle para guardar los tweets, cambiamos el atributo *max_id* para que se convierta en el tweet más antiguo que hemos descargado. Así, en la próxima iteración comenzará desde ahí y no repetirá tweets.

4.4.2 Limpieza de los datos

El siguiente paso a destacar dentro del procedimiento es el de la limpieza de los datos. Debemos preparar los datos a conciencia para que sirvan para nuestro propósito. Esta limpieza de datos consta de 2 pasos:

1. En primer lugar, queremos quedarnos solamente con la opinión de las personas, así que tenemos que eliminar todos los tweets que no hayan sido escritos por personas. Gracias a que Twitter mantiene información acerca de la fuente del tweet podemos realizar este procedimiento: Lo que hacemos es seguir los siguientes sub-pasos:

   a. Eliminamos el código *HTML*[17] para quedarnos simplemente con el campo *source,* que es el que necesitamos.

   b. Podemos deducir que los tweets escritos por personas (y no por bots o por sistemas de anuncios) están escritos desde dispositivos. Y por suerte Twitter especifica los dispositivos de la siguiente manera: '*Twitter <nombre del dispositivo>*'. De esta manera podemos encontrar: *Twitter for iPhone*, *Twitter for Android*, *Twitter for Windows*, *Twitter Web Client*, etc. Por lo tanto, ya que los nombres de los dispositivos empiezan por la palabra *Twitter*, eliminamos los demás. Sin embargo, quedará *Twitter Ads* que corresponde a los anuncios oficiales de Twitter, los cuales eliminaremos a mano.

[16] Los códigos de idiomas se establecen por la ISO 639-1

[17] Los tweets extraídos vienen con un formato similar a: <a href="https://ifttt.com" rel="nofollow">IFT...

c. El siguiente paso es opcional y depende de la búsqueda que queramos hacer. Se trata de eliminar términos dentro del texto que sepamos que no van a coincidir con la búsqueda que estamos realizando. Por ejemplo, si nuestra búsqueda se basa en encontrar tweets relacionados con la premier league inglesa, en este punto podríamos eliminar tweets que correspondan con Premier Leagues de otros países.

2. En segundo lugar, necesitamos preparar los datos textuales para que sean funcionales para los algoritmos de análisis de sentimientos. En este caso el procedimiento consta de tres pasos:

   a. **Tokenización:** La tokenización[18] es un término que se refiere separar o segmentar cada cadena de texto en palabras individuales. Para entenderlo correctamente podríamos decir que la tokenización transforma una cadena de texto como esta: "Ayer fui a la playa temprano !!!" en esto: *["Ayer", "fui", "a", "la", "playa", "temprano"," !!!"*].

   b. **Eliminar palabras reservadas:** Eliminar las palabras reservadas consta de eliminar todas aquellas palabras de uso común que un motor de búsqueda ha sido programado para ignorar: la, a, en, le, etc. Por lo tanto, la cadena anterior quedaría de la siguiente manera: *["Ayer", "fui", "playa", "temprano"," !!!"*].

   c. **Eliminar caracteres especiales:** Eliminar los caracteres especiales trata de eliminar los signos de puntuación, exclamación, interrogación, etc., que están dentro del texto. La cadena de ejemplo finalmente quedaría así: [*"Ayer", "fui", "playa", "temprano"*]

4.4.3 VADER

VADER (Valence Aware Dictionary and Sentiment Reasoner) es una herramienta de análisis de sentimientos basada en reglas que están específicamente adaptadas a los sentimientos expresados en las redes sociales. Se trata de una herramienta de código abierto. Dentro de VADER hay una amplia variedad de funcionalidades: Analizador de sentimientos, analizador de opiniones de películas, analizador de opiniones de Amazon, etc.

En nuestro caso utilizamos un analizador de sentimientos que los clasifica en negativos, neutrales o positivos. Este procedimiento lo hacemos para entender el funcionamiento generalizado de los algoritmos clasificadores y analizadores de sentimientos.

[18] El término hace referencia, en general a un sistema de seguridad en métodos de pago. Sin embargo, su traducción al inglés literal es simbología o tokenización. Como la simbología no se relaciona y el término hace referencia a dividir en tokens, utilizaremos tokenización.

La forma de funcionar de VADER es mediante el método *polarity_scores*, el cual recibe una cadena de texto y devuelve un vector donde se indica cuál es la puntuación que tiene cada sentimiento en esa cadena: *{'compound':0.7003,'neg':0.0,'neu':0.691,'pos':0.309}*

El valor *compound* es una puntuación normalizada de las tres categorías que está entre -1 y 1. Este valor está dado por la siguiente fórmula:

$$score_{norm} = \frac{x}{\sqrt{x^2 + \alpha}}, \alpha = 15$$

Como *compound* se sitúa entre -1 y 1 es lógico deducir que -1 se trata de un sentimiento puramente negativo, 0 de un sentimiento exclusivamente neutro y 1 un sentimiento únicamente positivo.

4.4.4 Análisis de resultados en los algoritmos

Una parte importante del proyecto es averiguar cuál es el mejor algoritmo. Para ello se necesitan establecer métodos que arrojen información útil acerca de lo eficientes que son los algoritmos. Estos métodos son los siguientes:

1. **Matriz de confusión:** Una matriz de confusión es una técnica para resumir el rendimiento de un algoritmo de clasificación. Proporciona información sobre lo que el modelo de clasificación está haciendo bien y qué tipos de errores está cometiendo. Lo que hace, concretamente, es dividir los resultados de pruebas en las siguientes categorías:
    a. **Verdadero positivo:** Valores predichos correctamente.
    b. **Falso positivo:** Valores predichos de manera incorrecta
    c. **Verdadero negativo:** Valores rechazados de manera correcta.
    d. **Falso negativo:** Valores rechazados de manera incorrecta.

    Usualmente, la manera de mostrar esta información es una matriz donde se ponen los datos de cada categoría:

Predichos/Reales	**Positivo**	**Neutro**	**Negativo**
Positivo	13	21	10
Neutro	2	12	8
Negativo	9	34	16

Tabla 2: Ejemplo de matriz de confusión

    A partir de esta información se pueden realizar una serie de mediciones para clasificar la eficiencia de los algoritmos:

    a. **Precisión:** La precisión se define como la proporción de predicciones positivas con respecto al número de observaciones que son

realmente positivas.

b. **Memoria:** Nos dice cuál es la proporción de observaciones realmente positivas que se predicen como positivas.

c. **F1-Score:** La puntuación F1 combina precisión y memoria para medir la precisión de la prueba. Puede ser interpretado como el promedio ponderado de precisión y memoria con su mejor valor en 1 y el peor en 0

d. **Soporte:** El soporte es el número de observaciones que se predicen en una clase en particular. Para que los datos de prueba sean correctos, deberían estar equilibrados, es decir, que existan más o menos la misma cantidad de negativos, positivos y neutros.

La manera de visualizar esta información es en forma de tabla, aquí tenemos un ejemplo con Naive Bayes:

Naïve Bayes	Precisión	Memoria	Puntuación F1	Soporte
Negativo	0.80	0.50	0.62	8
Neutro	1.00	0.20	0.33	5
Positivo	0.20	0.67	0.31	3
Media/Total	0.75	0.44	0.47	16

Tabla 3: Ejemplo de tabla con Naive Bayes

2. **Validación cruzada:** Los datos de entrada se dividen en K partes, una de las cuales está reservada para pruebas y la otra K-1 para entrenamiento. El proceso descrito se repite K veces y se promedian las métricas de evaluación. Esto ayuda a determinar cómo de bien un modelo se generalizaría a nuevos conjuntos de datos. El resultado de esta prueba se reduce a un número entre 0 y 1; cuanto más se acerque a 1, hará mejores predicciones en nuevos modelos de datos.

4.5 Algoritmos utilizados

Dentro del análisis de algoritmos, nuestro proyecto utiliza concretamente cuatro clasificadores distintos: *SVC, MLP, KNN y NB*. A continuación, vamos a explicar en detalle los parámetros utilizados para mejorar el rendimiento de tres de estos clasificadores, no sin antes mencionar que la complejidad de seleccionar correctamente estos parámetros es muy alta, debido a que cada muestra de entrenamiento es distinta. Las pruebas con diferentes valores harán que se pueda escoger un mejor ajuste.

4.5.1 SVC. Support Vector Classification

Es un tipo de clasificador que separa las clases en dos (o más) espacios de un plano, dentro de este plano, cuanto más separadas estén las clases mejor será el funcionamiento. El vector de soporte (que divide ambos espacios) es el que determina en que clase debe estar una muestra nueva.

Este clasificador dispone de muchos parámetros, en nuestro caso hemos atendido a los siguientes principales:

- **kernel:** (*kernel = 'string'*) Especifica el tipo de núcleo usado por el algoritmo. Puede ser *'linear', 'poly', 'rbf', 'sigmoid' o 'precomputed'*. Por defecto se usa *rbf.*

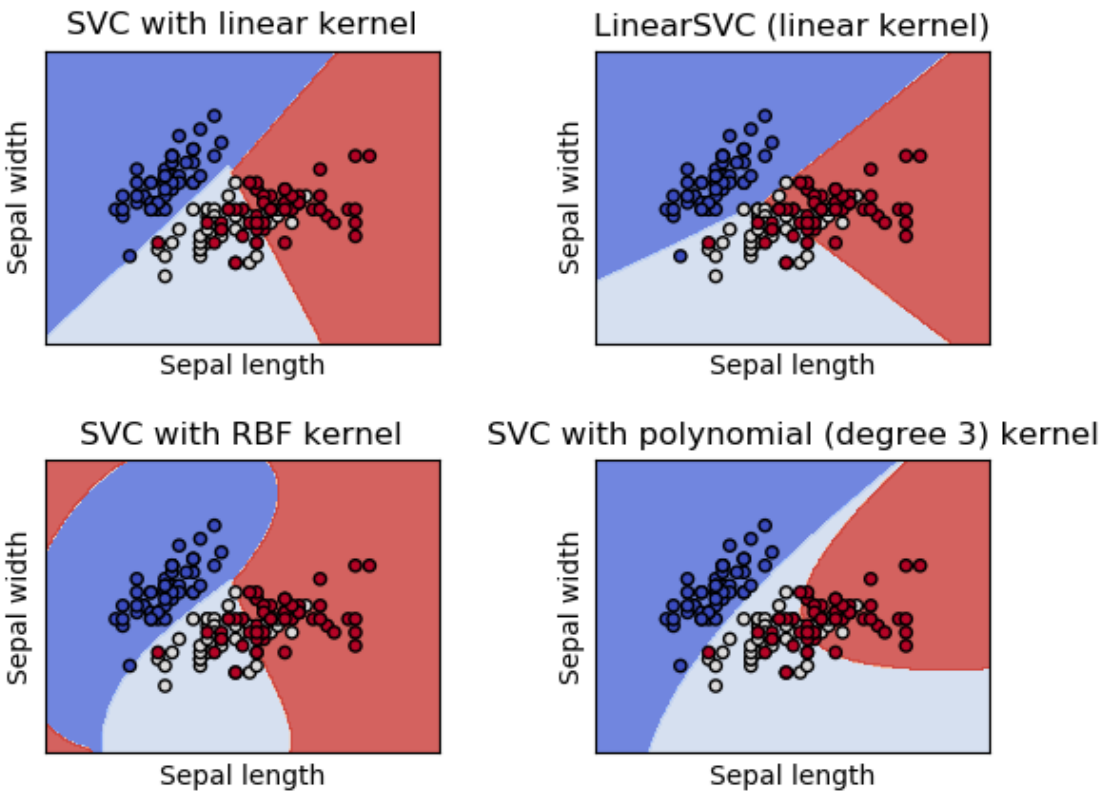


Ilustración 1: Tipos de Kernel en SVC

- **degree:** (*degree = int*) Grado de la función cuando el núcleo es de tipo *poly*. (Como se puede ver en la ilustración 1) Se ignora en los demás casos. Por defecto es 3.
- **gamma:** (*gamma = float*) Coeficiente para *rbf, poly y sigmoid*. Si es auto, que es el establecido por defecto, es igual a *1/número de tweets*. Su uso es para determinar cuánta influencia tiene un solo ejemplo de entrenamiento, cuanto más grande es gamma, más cerca deben verse otros ejemplos para ser afectados.

    - **C:** (c = float) Otro parámetro importante para el rendimiento de SVC es C, el cual intercambia la clasificación errónea de ejemplos de entrenamiento con la simplicidad de la superficie de decisión. En resumen y para entenderlo, un nivel alto clasifica correctamente todos los ejemplos de entrenamiento, mientras que un nivel bajo es *más permisivo*.

- **coef0:** (*coef0 = float*) Término independiente para *poly* y *sigmoid*.
- **probability:** (*probability = True/False*) Indica si se permiten o no las estimaciones de probabilidad. Debe activarse antes de llamar al método fit[19].
- **Cache_size:** (cache_size = float) indica el tamaño de la caché del Kernel.
- **shrinking:** (*shrinking = True/False*) Indica si se va a usar la heurística retráctil o no.

Existen, no obstante, ciertas recomendaciones con son recomendables tener en cuenta a la hora de establecer estos datos:

1. **Evitar copiar los datos:** Esta recomendación se basa en que cuando se copia los datos entre distintos tipos de flotantes con distinta precisión se puede perder información. Se recomienda utilizar precisión doble y las librerías de numpy.
2. **El tamaño de la caché del Kernel:** Se recomienda que el tamaño de la caché sea de 500 o 1000MB
3. **Escoger un buen C:** C es 1 por defecto, y puede ser razonable y lógico. Pero si hay muchas observaciones que se consideren *ruidosas* se debe disminuir.
4. **Desequilibrio de datos:** Si los datos para clasificar están desequilibrados (muchos positivos y pocos negativos, por ejemplo) es conveniente establecer *class_weight = 'balanced'*[20]

Por último, para tener un conocimiento más amplio del algoritmo, es importante tener en cuenta que su complejidad se relaciona con un problema de programación cuadrática y que por tanto, el solucionador de dicho problema implementado para SVC se escala entre: $O(n_{features} \times n_{samples}^2)$ y $O(n_{features} \times n_{samples}^3)$.

[19] Este método se encarga de ajustar los datos a cada modelo en concreto. Es necesario utilizar este método antes de entrenar los algoritmos, ya que cada algoritmo lleva un ajuste concreto.

[20] Como se puede suponer, este parámetro indica al modelo que deben balancearse los datos, por lo que iniciará un procedimiento de balanceo previo al entrenamiento.

4.5.2 MLP. Multi-Layer Perceptron Classifier

Se trata de una red neuronal artificial de alimentación anticipada[21]. Está formada por tres capas de neuronas (nodos) que tienen una función de activación no lineal. Tiene la capacidad de resolver problemas que no son linealmente separables.

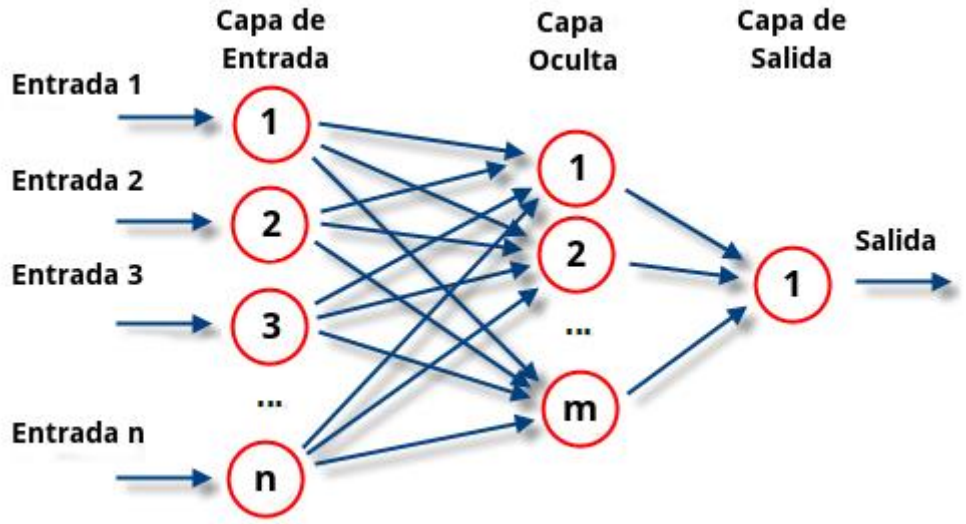


Ilustración 2: Desglose de un MLP

Veamos cómo funciona en el entrenamiento: Se ejecuta un algoritmo denominado de retropropagación[22] del error que consiste en lo siguiente: Se introduce en la red vectores de características cuya clase, y por tanto su salida, es conocida. El vector de características es procesado produciendo un resultado en la capa de salida. Esta salida es comparada con la salida deseada, es decir, la que nosotros hayamos especificado por pertenecer a cierta clase. El error entre la salida real y la deseada se realimenta hacia atrás por las capas, para variar los valores de los pesos con el objetivo de ir minimizando el error de en cada iteración del proceso de entrenamiento.

Este clasificador tiene multitud de opciones de configuración mediante parámetros. A continuación, se explican los más importantes.

- **hidden_layer_sizes:** (*hidden_layer_sizes = (int,))* Indica el número de neuronas en la capa oculta. Por defecto es 100.
- **activation:** *(activation=string)* Especifica la función de activación para la capa oculta. Puede ser:
    - **'identity':** Sin operación, útil para implementar cuellos de botella lineales.

[21] O de alimentación directa (FeedForward), se trata del tipo más simple de red neuronal en la cual la información se mueve en una sola dirección. Más información.

[22] O Backpropagation: Utilizada por redes con más de una capa oculta para el gradiente (derivadas de múltiples variables) necesario para el cálculo de los pesos que se utilizarán en la red.

- **'logistic':** Función sigmoide logística.
- **'tanh':** Función de la tangente hiperbólica
- **'relu':** Función de la unidad lineal rectificada (Por defecto)

- **alpha:** (alpha = float) Parámetro de penalización L2 (plazo de regularización). por defecto es 0.0001.
- **learning_rate:** (learning_rate = string) Es la manera en la que se define la tasa de aprendizaje. Puede ser:
  - **'constant':** Tasa de aprendizaje constante.
  - **'invscaling':** Disminuye gradualmente la velocidad de aprendizaje
  - **'adaptative':** Mantiene el ritmo de aprendizaje constante siempre y cuando la pérdida de información siga disminuyendo. Si en dos ocasiones consecutivas no se reduce la pérdida de entrenamiento o no se aumenta la puntuación de validación el ritmo de aprendizaje se divide por 5.

4.5.3 KNN. K-Neighbors Classifier

Clasificador que implementa el voto de los k-vecinos más cercanos. Estima la función de densidad por cada clase. Es decir, se basa en averiguar la probabilidad de que un elemento pertenezca a una clase. Para averiguar dicha probabilidad se calcula un voto de mayoría simple de los vecinos más cercanos a cada punto.

Al igual que los anteriores clasificadores, KNN dispone de muchas opciones de personalización para hacer el entrenamiento lo más eficiente posible. Aquí se muestran los parámetros más importantes:

- **n_neighbors:** (n_neighbors = int) Número de vecinos a usar para las consultas.
- **weights:** (weights = str) Función de peso usada para las predicciones. Pueden ser:
  - **'uniform':** Pesos uniformes. Todos los puntos en todo el vecindario son iguales.

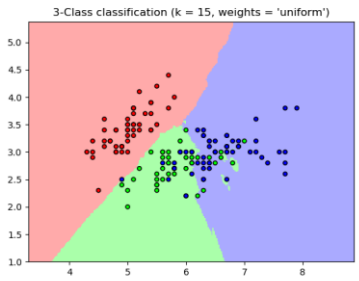


Ilustración 3: KNN con pesos uniformes

- **'distance':** Los vecinos más cercanos de un punto de consulta tendrán una mayor influencia que los vecinos que están más lejos.

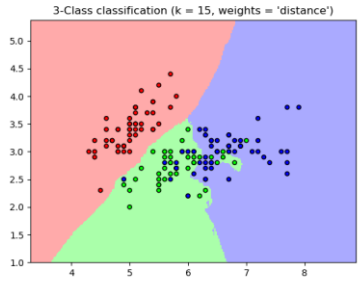


Ilustración 4: KNN con pesos dependientes de las distancias

- **algorithm:** (algorithm = string) Algoritmo usado para los vecinos más cercanos. Pueden ser:
  - **'brute':** usa una búsqueda de fuerza bruta de las distancias entre todos los pares de puntos en el conjunto de datos. Resulta una buena elección para muestras de datos pequeñas. Sin embargo conforme crece la muestra, se hace inviable debido a la escala $O[DN^2]$
  - **'kd_tree':** Usa *kd_tree* como solución a la ineficiencia de la fuerza bruta con N grande, para ello se implementa una estructura en árbol. La idea tras este algoritmo es que si el punto A es muy distante del punto B y el punto B está muy cerca del punto C, entonces sabemos que los puntos A y C son muy distantes sin tener que calcular su distancia. Esto permite reducir la escalabilidad a $O[DN\log(N)]$ donde la D son las dimensiones (clases) y N el número de muestras.
  - **'ball_tree':** Usa *ball_tree* como solución a la ineficiencia de *kd_tree* con D grande. *Kd_tree* utilizaba árboles binarios que dividían recursivamente el espacio de parámetros a lo largo de los ejes de datos, esto implica que se construyan muy rápidamente. Sin embargo, *ball_tree* construye los árboles dividiendo los datos en una serie de hiperesferas. Esto hace que la construcción sea más lenta, pero mucho más eficiente en su utilización, incluso con dimensiones altas.
  - **'auto':** intentará decidir el algoritmo más apropiado basándose en los valores pasados para el método de ajuste.
- **leaf_size:** (leaf_size=int) El tamaño de la hoja usado por KDTree. Afecta a la velocidad de la construcción, a la consulta y a la memoria necesaria para almacenar el árbol. El valor óptimo depende de la naturaleza del problema.
- **p:** (p = int) Parámetro de potencia para la métrica de Minkowski. puede ser p=1 para usar distancia_manhattan y p=2 para usar distancia_minkowski

En este caso, incluso más que en los anteriores casos, la elección del algoritmo es

una tarea compleja, ya que depende mucho de cómo se dimensionan las muestras y la estructura de los datos.

4.6 Elementos del prototipo y funcionamiento

En el caso de este proyecto, los elementos del prototipo son exclusivamente software y cada uno de ellos juega un papel indispensable para el correcto funcionamiento y usabilidad requeridas en el prototipo. Los elementos que se describen a continuación son partes bien diferenciadas del código, y juntas engloban toda la funcionalidad. Vamos a explicar cómo se implementa cada una de ellas y que funcionamiento tienen, tanto en el código como en su interfaz gráfica.[23]

4.6.1 Entrenamiento de algoritmos

Pseudocódigo

Solicitar **ficheros** de entrenamiento y test
Realizar la **tokenización** de los dos conjuntos de datos
Transformar los datos para el entrenamiento
Por cada algoritmo:
    Por cada tipo de parámetro:
        Por cada posibilidad de parámetro:
            Entrenar el algoritmo
            Si la puntuación del algoritmo es la mayor:
                Almacenar la puntuación del modelo
Guardar el mejor modelo de los 4 existentes
Entrenar los algoritmos con los parámetros de las mejores puntuaciones

Detalles de la implementación

El entrenamiento de los algoritmos es la sección del programa que, recibiendo un conjunto de datos, averigua cuál es la configuración de cada algoritmo que obtiene un mejor resultado y la guarda.

Esta función se llama en el momento en el que se hace clic al botón de la interfaz de entrenar algoritmos:

```
# Conectar a entrenar_algoritmos
self.buttonEntrenar.clicked.connect(self.entrenar_algoritmos)
```

La función de entrenar algoritmos sigue los siguientes pasos:

1. En primer lugar, se solicitan los datos de entrenamiento y de pruebas, que se encuentran en ficheros, para poder entrenar los algoritmos, a su vez se carga la barra de progresos para poder visualizar el progreso de entrenamiento.

[23] Explicaremos en este apartado el código que interesa para la implementación del análisis de sentimientos, no indagaremos en el código perteneciente a la interfaz gráfica.

```
self.progressBarUnTweet.reset()
self.progressBarUnTweet.setMaximum(438)
self.progressBarUnTweet.setMinimum(0)
filename2 = self.dialogo1.getOpenFileName(self, "Selecciona el fichero de entrenamiento","/")
filename2 = filename2[0].split("/")
filename2 = filename2[-1]
filename = self.dialogo1.getOpenFileName(self, "Selecciona el fichero de pruebas", "/")
filename = filename[0].split("/")
filename = filename[-1]
dataset = pd.read_csv(filename)
tweetys = dataset['text']
dataset2 = pd.read_csv(filename2)
```

2. A continuación, se realiza la limpieza de los datos, tanto del dataset1 donde se encuentran los datos de entrenamiento, como del dataset2 donde están los datos de test.

```
#LIMPIEZA DE DATOS DE DATASET
#TOKENIZATION
dataset['tokens'] = dataset['text'].apply(TweetTokenizer().tokenize)
#STOPWORDS
stopwords_vocabulary = stopwords.words('english') #estará en español?
dataset['stopwords'] = dataset['tokens'].apply(lambda x: [i for i in x if i.lower() not in stopwords_vocabulary])
#SPECIAL CHARACTERS AND STOPWORDS REMOVAL
punctuations = list(string.punctuation)
dataset['punctuation'] = dataset['stopwords'].apply(lambda x: [i for i in x if i not in punctuations])
dataset['digits'] = dataset['punctuation'].apply(lambda x: [i for i in x if i[0] not in list(string.digits)])
dataset['final'] = dataset['digits'].apply(lambda x: [i for i in x if len(i) > 1])
print("2. Limpieza del dataset realizada")
self.progressBarUnTweet.setValue(1)
```

3. En tercer lugar, se prepara el vector de entrenamiento. En este punto, la variable vectorizer es almacenada en un fichero para posteriores entrenamientos. Hay que tener en cuenta que esta variable no es propia de la clase ya que necesita ser utilizada en muchas de las funciones de manera global y siempre con el mismo valor, de hecho, es inicializada en el programa principal donde se realiza la comprobación; si existe un archivo con esa variable guardada lo abre, y si no, la crea en ese momento.

```
train_data = dataset2['final'][0:500]
train_labels = dataset2['label'][0:500]

test_data = dataset['final'][0:125]
test_labels = dataset['label'][0:125]

train_data = list(train_data.apply(' '.join))
test_data = list(test_data.apply(' '.join))
self.progressBarUnTweet.setValue(3)
self.progresLabel.setText("Actualizando datos de entrenamiento")
```

```
train_vectors = self.vectorizer.fit_transform(train_data)
test_vectors = self.vectorizer.transform(test_data)

fvecto = open('fvecto', 'wb')
dump(self.vectorizer, fvecto)

modelos = ['NaiveBayes','Svc','Knn','Mlp']
```

A continuación, la función condicional del programa principal para la creación del vector de entrenamiento:

```
if  path.exists('fvecto'):
    fvecto = open('fvecto', 'rb')
    vectorizer = load(fvecto)
else:
    vectorizer = TfidfVectorizer(min_df=5, max_df = 0.8, sublinear_tf=True, use_idf=True)
```

4. El siguiente paso es entrenar los algoritmos y almacenar cuál es el mayor resultado para cada configuración de cada uno de los algoritmos. Por motivos de espacio, sólo vamos a mostrar el código relativo al algoritmo SVC.

```
puntuaciones = [0,0,0,0]
params_svc = [['linear','poly','tbf','sigmod','precomputed'],[3,5,10],[0.1,0.5,0.9],[True,False],[True,False]]
best_svc = []
params_knn = [[1,5,10],['uniform','distance'],['ball_tree','kd_tree','brute','auto'],[5,30,100],[1,2]]
best_knn = []
params_mlp = [[50,100,150],['identity','logistic','tanh','relu'],[0.00005,0.0001,0.001],['constant','invscaling','adaptative']]
best_mlp = []
self.progressBarUnTweet.setValue(4)
self.progresLabel.setText("Preparando parámetros de algoritmos")
#ENTRENAMIENTO DE ALGORITMOS
progreso = 5
for alg in modelos:
    if alg == 'Svc':
        for a in params_svc[0]:
            for b in params_svc[1]:
                for c in params_svc[2]:
                    for d in params_svc[3]:
                        for e in params_svc[4]:
                            mod = SVC(kernel=a,degree=b,coef0=c,probability=d,shrinking=e)
                            punt = self.entrenar(alg,train_vectors,train_labels,test_vectors, test_labels, mod)
                            self.progressBarUnTweet.setValue(progreso)
                            progreso = progreso + 1
                            self.progresLabel.setText("Entrenando SVC con kernel "+a)
                            if punt > puntuaciones[0]:
                                puntuaciones[0] = punt
                                best_svc = [a,b,c,d,e]
```

5. A continuación, una vez tengamos todos los datos almacenados en el vector de puntuaciones, vamos a establecer cuál es la mejor configuración para después entrenar los algoritmos con la mejor configuración posible.

```
tmp = 0
guia = 0
for h in puntuaciones:
    if h > tmp:
        best_model = guia
        tmp = h
    guia = guia + 1
self.progressBarUnTweet.setValue(progreso)
progreso = progreso + 1
self.entrenar(alg,train_vectors,train_labels,test_vectors, test_labels,
SVC(kernel=best_svc[0],degree=best_svc[1],coef0=best_svc[2],probability=best_svc[3],shrinking=best_svc[4]))
self.entrenar(alg,train_vectors,train_labels,test_vectors, test_labels, mod = MultinomialNB())
self.entrenar(alg,train_vectors,train_labels,test_vectors, test_labels,
KNeighborsClassifier(n_neighbors=best_knn[0],weights=best_knn[1],algorithm=best_knn[2],leaf_size=best_knn[3],p=bes
t_knn[4]))
self.entrenar(alg,train_vectors,train_labels,test_vectors, test_labels,
MLPClassifier(hidden_layer_sizes=best_mlp[0],activation=best_mlp[1],alpha=best_mlp[2],learning_rate=best_mlp[3]))
```

6. La función entrenar se encarga de realizar el entrenamiento propiamente dicho y se define de la siguiente manera:

```
def entrenar(self, alg,train_vectors,train_labels,test_vectors, test_labels, mod):
    nfile = alg
    if path.exists(nfile):
        file = open(nfile, 'rb') #abre el archivo en modo lectura
        mod = load(file) #carga el archivo en la variable
        mod.fit(train_vectors, train_labels).score(test_vectors, test_labels) #lo entrena
        file.close() #cierra el archivo
        file = open(nfile, 'wb') #abre el archivo en modo escritura
        dump(mod, file) #actualiza el entrenamiento
    else:
        file = open(nfile, 'wb') #abre el archivo en modo escritura
        mod.fit(train_vectors, train_labels).score(test_vectors, test_labels) #lo entrna
        dump(mod, file) #guarda el entrenamiento
    predicted = cross_val_predict(mod, test_vectors, test_labels, cv=10)
    return accuracy_score(test_labels,predicted)
```

Funcionamiento de interfaz gráfica

En cuanto a la interfaz gráfica, el entrenamiento se puede seleccionar en el botón que se sitúa en la esquina superior izquierda de la pantalla principal.

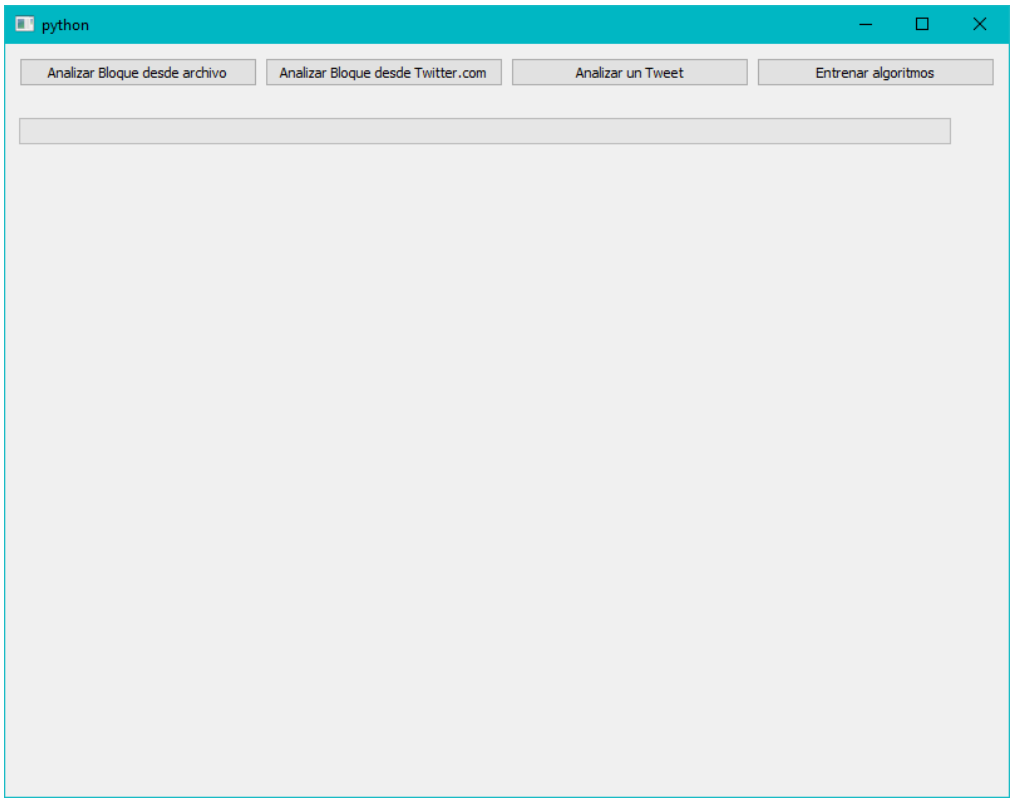


Ilustración 5: Ventana principal

Al pulsar este botón se activa la función de entrenar algoritmos y comienzan a solicitarse los datos de entrenamiento:

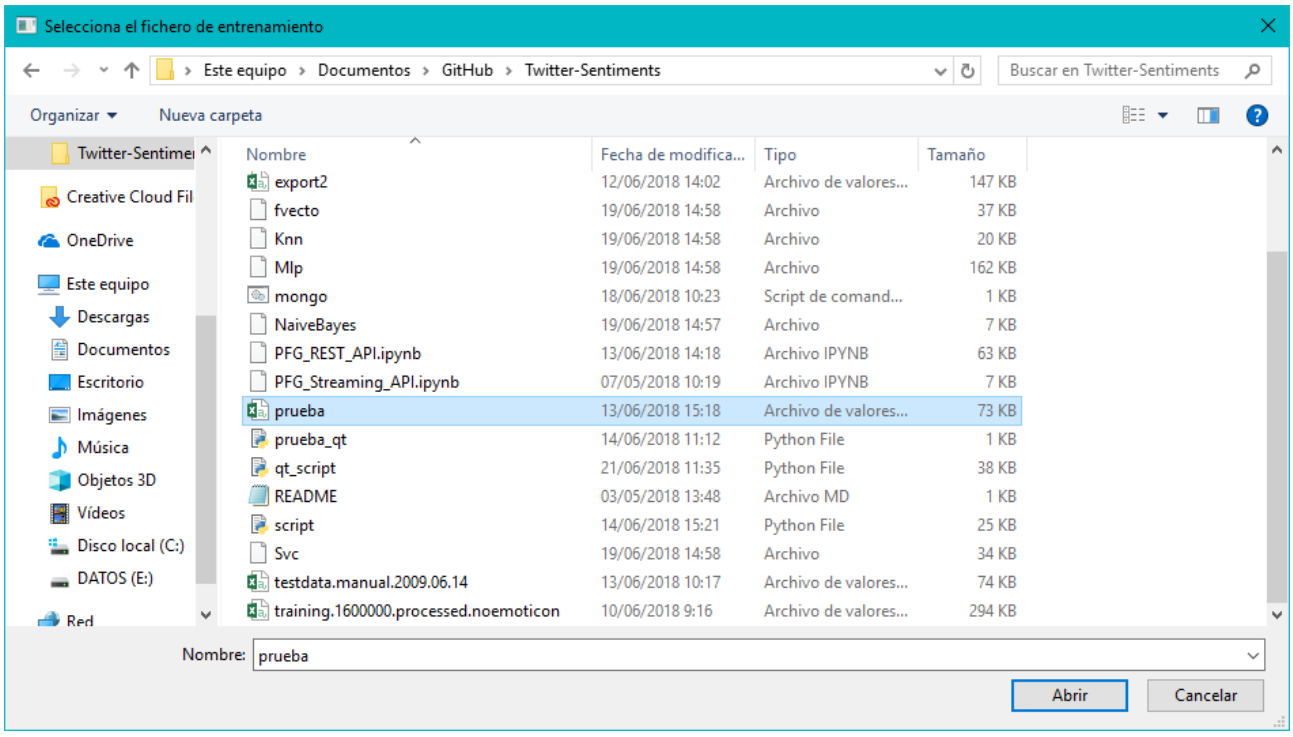


Ilustración 6: Ventana de selección de fichero de datos de entrenamiento

Así como los datos de test:

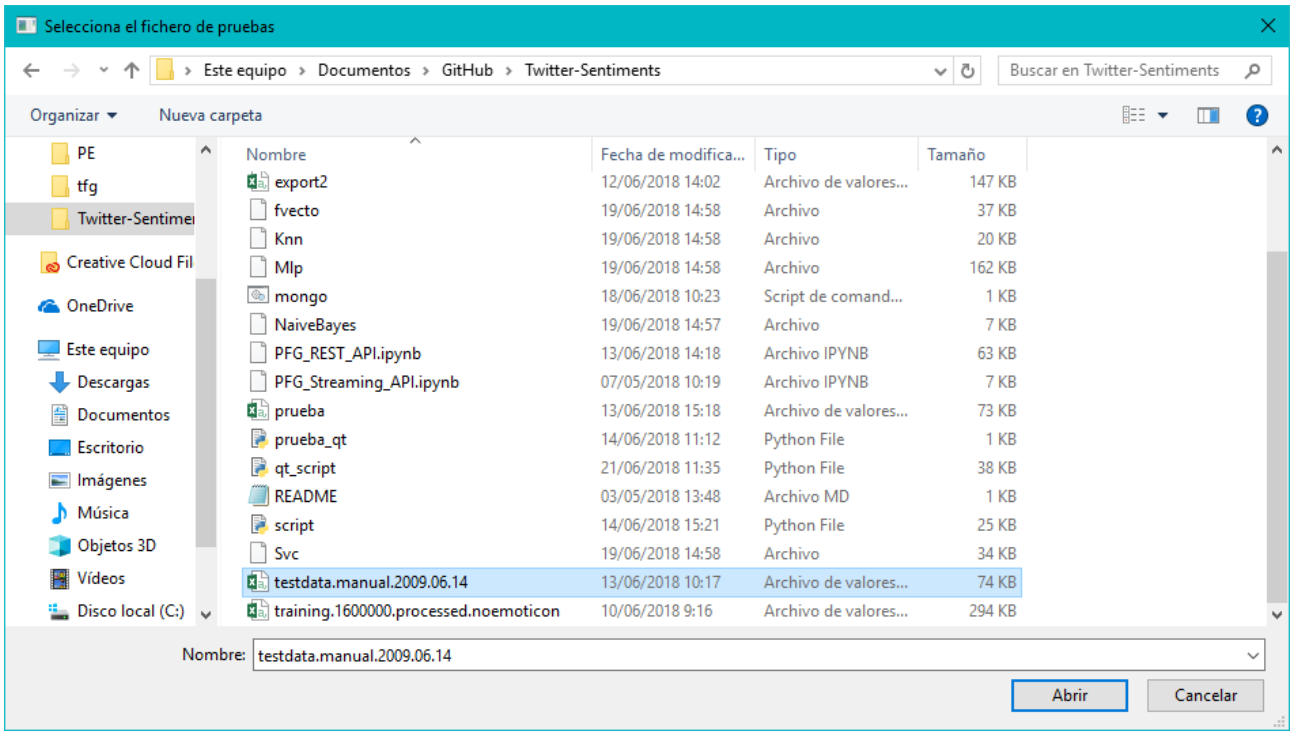


Ilustración 7: Ventana de selección de fichero de datos de pruebas

Una vez introducida la información, el programa se pone a trabajar y a probar todas las configuraciones posibles para obtener la mejor configuración de cada algoritmo. Mientras se realiza el proceso, la barra de progresos va indicando cuanto trabajo queda por realizar. Debido a que en algunos de los procesos se requiere mucho tiempo, es importante informar al usuario de que la aplicación está funcionando mientras se espera.

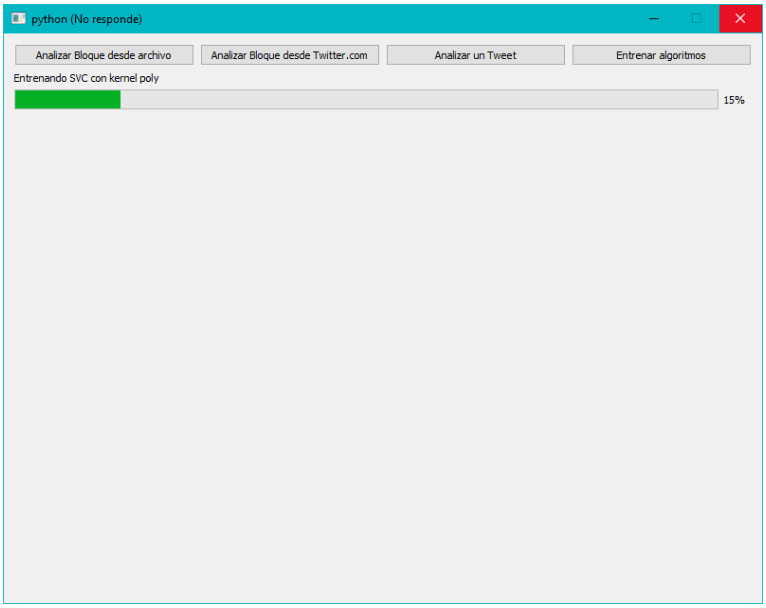


Ilustración 8: Ventana principal durante el entrenamiento de los algoritmos

4.6.2 Analizar un tweet individual

Pseudocódigo

Establecer **claves de acceso** para twitter
Declarar parámetros de **consulta**
Inicializar la **base de datos**
Por cada página:
    Solicitar 1 tweet a Twitter
    Almacenar los tweets en la base de datos
    Guardar el id del último tweet (para paginación)
Guardar los datos recogidos de la base de datos en un **dataframe**
Realizar la **limpieza de datos**
Realizar la **Tokenización** de los datos
Realizar el reconocimiento de entidades con **NER**
Predecir los resultados de los sentimientos por cada algoritmo
Mostrar la información en la ventana principal

Detalles de la implementación

Otra sección importante del programa es el procedimiento por el cual se extrae de Twitter un único tweet y se analiza de manera individual para obtener datos acerca de los sentimientos encontrados en el texto del tweet, así como información concerniente a entidades encontradas en ese tweet.

La función de analizar un solo tweet se llama de manera similar a cuando se quiere entrenar un algoritmo:

```
self.buttonUnTweet.clicked.connect(self.analizarUnTweet)
```

La función *analizarUnTweet* se define de la siguiente manera:

1. En primer lugar, se establecen las claves de acceso para la API de twitter y se definen las variables que almacenan la información referente a las búsquedas que se quieren realizar. Además de eso, previamente, se solicita al usuario el contexto sobre el cual se quiere buscar un tweet.

```
self.consultaText = self.dialogConsulta.getText(self,"Consulta de Twitter", "¿Sobre qué quieres buscar?")
self.progressBarUnTweet.reset()
self.progressBarUnTweet.setMaximum(10)
self.progressBarUnTweet.setMinimum(0)
consumer_key='ynSB0dFvqPl3xRU7AmYk39rGT'
consumer_secret='6aIIXTKSxf0RE57QK3fDQ8dxdvlsVr1IRsHDZmoSlMx96YKBFD'
access_token='966591013182722049-BVXW14Hf5s6O2oIwS3vtJ3S3dOsKLbY'
access_token_secret='829DTKPjmwsSytmp1ky9fMCJkjV0LZ04TbL9oqHGV6cDm'
#parámetros de la consulta
q = self.consultaText[0] + ' -filter:retweets AND -filter:replies'
url = 'https://api.Twitter.com/1.1/search/tweets.json'
pms = {'q' : q, 'count' : 1, 'lang' : 'en', 'result_type': 'recent'}
auth = OAuth1(consumer_key, consumer_secret, access_token,access_token_secret)
```

2. A continuación, lo que se hace es inicializar la base de datos para poder almacenar la información. Este procedimiento de guardar el tweet en la base de datos podría parecer una pérdida de tiempo a priori, pero con el fin de ir recopilando tweets para el posterior etiquetado de cualquier tema en concreto, siempre que se descargue información de twitter, se procede a almacenar en una base de datos.[24]

```
database_name = "baseDeDatos"
collection_name = "coleccion"
client = MongoClient('mongodb://localhost:27017/')
db = client[database_name]
collection = db[collection_name]
```

3. El siguiente paso es el de la paginación. En twitter existen dos maneras de descargar la información que necesitamos; por medio del *REST API* o por medio del *Streaming API*. En el segundo caso, como su nombre indica, se obtienen tweets de Twitter conforme sean posteados en la red social. Es bastante útil cuando se pretende realizar análisis de las cosas que van sucediendo en tiempo real. Podría parecer el método más lógico para sacar un solo tweet de Twitter, pero el punto negativo de *Streaming API* es que su periodo de acción es de 7 días, es decir, si nadie ha *twiteado* un tweet del tema seleccionado en un plazo de 7 días, no encontraremos nada y no se realizará el análisis.[25] REST API sin embargo obtiene un histórico de todos los tweets que se relacionen con la búsqueda deseada, por lo tanto sólo necesitamos indicarle que cantidad de tweets deseamos descargar (uno en nuestro caso[26]) y buscará el último tweet del histórico. El método de la paginación será explicado en la función de analizar un bloque de tweets, que es donde realmente se utiliza.

```
pages_counter = 0
number_of_pages = 1 #Número de veces que cuenta
   while pages_counter < number_of_pages:
     pages_counter += 1
     res = requests.get(url, params = pms, auth=auth)
     tweets = res.json()
     ids = [i['id'] for i in tweets['statuses']]
     pms['max_id'] = min(ids) - 1
     collection.insert_many(tweets['statuses'])
```

4. En cuarto lugar, los datos obtenidos y almacenados en la base de datos se

[24] Esta decisión viene dada por la idea de poder mejorar la aplicación en futuros desarrollos.

[25] La aplicación se quedará *colgada* esperando a encontrar un tweet, o en este caso, esperando a que alguien twitee.

[26] Sin embargo, después de varias pruebas, lo recomendable es solicitar más de uno, ya que suele dar errores al solicitar un solo tweet. Recordemos que el valor que viene por defecto es 15.

pasan a un *dataframe*[27] y se limpian según el procedimiento que vimos en el caso del entrenamiento de los algoritmos. En este caso, al tratarse de datos de twitter se llama a una función que se implementa de la siguiente manera:

```
def limpieza_de_datos_de_twitter(self, df):
    df['tweet_source'] = df['source'].apply(lambda x: BeautifulSoup(x).get_text())
    devices =
list(set(df[df['tweet_source'].str.startswith('Twitter')]['tweet_source']))
    devices.remove('Twitter Ads')
    df = df[df['tweet_source'].isin(devices)]
    return df
```

5. También el Tokenizado se realiza mediante una función llamada *tokenizar*, similar a la que teníamos en el entrenamiento de los algoritmos:

```
def tokenizar(self, df):
    #TOKENIZATION inicial para NER
    df['tokens'] = df['text'].apply(TweetTokenizer().tokenize)
    #STOPWORDS
    stopwords_vocabulary = stopwords.words('english')
    df['stopwords'] = df['tokens'].apply(lambda x: [i for i in x if i.lower() not
in stopwords_vocabulary])
    #SPECIAL CHARACTERS AND STOPWORDS REMOVAL
    punctuations = list(string.punctuation)
    df['punctuation'] = df['stopwords'].apply(lambda x: [i for i in x if i not in
punctuations])
    df['digits'] = df['punctuation'].apply(lambda x: [i for i in x if i[0] not in
list(string.digits)])
    df['final'] = df['digits'].apply(lambda x: [i for i in x if len(i) > 1])
    return df
```

6. A continuación, vienen los dos pasos principales de la función, se realiza la llamada a una función declarada como: *usar_NER* que se encarga de realizar el reconocimiento de entidades. Esta función se detalla de la siguiente manera:

```
def usar_NER(self, tweetys, n):
    #st = StanfordNERTagger(r'C:\Users\Servicio Técnico\Documents\stanford-ner-
2018-02-27\classifiers\english.all.3class.distsim.crf.ser.gz')[28]
    st = StanfordNERTagger('/Users/jonas/stanford-ner-2018-02-
27/classifiers/english.all.3class.distsim.crf.ser.gz')
```

[27] Según su descripción: Estructura tabular bidimensional de datos potencialmente heterogéneos con ejes etiquetados, es decir, una especie de diccionario.

[28] Este comentario define el lugar donde se encuentra la instalación del NER de Stanford para el ordenador con sistema operativo Windows donde también realizaba el desarrollo del proyecto.

```
    entities = []
    tindice = 0
    for r in tweetys:
        lst_tags = st.tag(r)
        for tup in lst_tags:
            self.progressBarUnTweet.setValue(tindice)
            tindice = tindice + 1
            if(tup[1] != 'O'):
                entities.append(tup)
    df_entities = pd.DataFrame(entities)
    self.progressBarUnTweet.setValue(len(tweetys)*10)
    if df_entities.size >0:
        df_entities.columns = ["word","ner"]
        #Organizaciones
        organizations =df_entities[df_entities['ner'].str.contains("ORGANIZATION")]
        cnt = Counter(organizations['word'])
        organizaciones = cnt.most_common(n)
        #Personas
        person =df_entities[df_entities['ner'].str.contains("PERSON")]
        cnt_person = Counter(person['word'])
        personas = cnt_person.most_common(n)
        #Localizaciones
        locations =df_entities[df_entities['ner'].str.contains("LOCATION")]
        cnt_location = Counter(locations['word'])
        lugares = cnt_location.most_common(n)
        return (organizaciones, personas, lugares)
    else:
        return 0
```

7. El siguiente paso es utilizar los distintos algoritmos clasificadores y almacenar el resultado en una variable para, más tarde, mostrarlo en la interfaz gráfica.

```
nb = self.predecir_Naive_Bayes(test_vectors)
svc = self.predecir_SVC(test_vectors)
knn = self.predecir_KNN(test_vectors)
mlp = self.predecir_MLP(test_vectors)
```

8. Los siguientes pasos tratan de mostrar la información obtenida en tablas según los resultados obtenidos en NER y en los algoritmos clasificadores de sentimientos. Se trata de sencillos condicionales donde se analiza el resultado de los algoritmos (que sería 0, 2 o 4) y se transforman a texto (negativo, neutro o positivo). Un ejemplo para el clasificador Naive Bayes sería el siguiente:

```
if nb[1][0] == 1:
    self.tablaSent.setItem(0,1,QTableWidgetItem("Positivo"))
elif nb[1][1] == 1:
```

```
    self.tablaSent.setItem(0,1,QTableWidgetItem("Neutro"))
elif nb[1][2] == 1:
    self.tablaSent.setItem(0,1,QTableWidgetItem("Negativo"))
self.tablaSent.setItem(1,0,QTableWidgetItem("Clasificador SVC"))
```

Funcionamiento de interfaz gráfica:

En cuanto a la interfaz gráfica, el análisis de un solo tweet se puede seleccionar en la pantalla principal, al igual que el resto de opciones, en este caso es la segunda opción comenzando por la derecha.

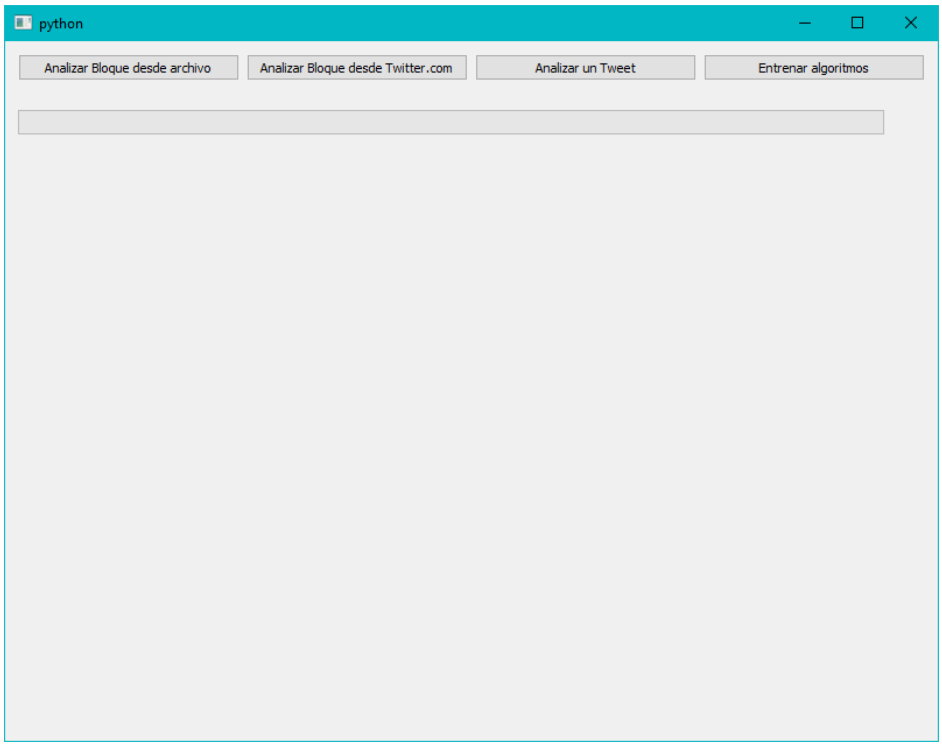


Ilustración 9: Ventana principal

Cuando se pulsa este botón se llama a la función correspondiente y aparece una ventana que solicita información referente a qué temática se va a buscar en Twitter para realizar el análisis.

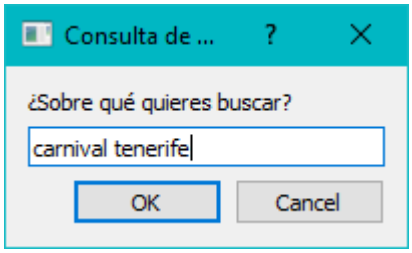


Ilustración 10:Ventana de selección del tema a buscar

Cuando se selecciona el texto (preferiblemente en inglés) [29]comienza todo el proceso anteriormente descrito por el cual se determina tanto las entidades que tiene ese tweet en particular como también los sentimientos encontrados para cada algoritmo clasificador.

Ilustración 11: Ventana principal con los datos del análisis de un tweet individual

[29] La búsqueda está parametrizada con el lenguaje en inglés por defecto. Esto no implica que no vaya a encontrar resultados en español, pero encontrará muchos menos.

4.6.3 Analizar un Bloque de Tweets de Twitter.com

Pseudocódigo

Establecer **claves de acceso** para twitter
Declarar parámetros de **consulta**
Inicializar la **base de datos**
Por cada página:
    Solicitar 1 tweet a Twitter
    Almacenar los tweets en la base de datos
    Guardar el id del último tweet (para paginación)
Guardar los datos recogidos de la base de datos en un **dataframe**
Realizar la **limpieza de datos**
Realizar la **Tokenización** de los datos
Realizar el reconocimiento de entidades con **NER**
Por cada entidad reconocida:
    Realizar el análisis de sentimientos por cada algoritmo
    Crear un gráfico que muestra los resultados

Detalles de la implementación

En el caso que se desee analizar un bloque de archivos extraídos directamente de Twitter.com los pasos a seguir son muy similares a analizar un solo tweet. En este punto analizaremos las diferencias:

1. Es importante tener en cuenta en el primer punto, que en el caso de analizar un bloque de Tweets se utilizan variables de clase que están definidas previamente mediante la función *cuadroDialogo*, que se muestra a continuación.

```
def cuadroDialogo(self):
   self.consultaText = self.dialogConsulta.getText(self,"Consulta de Twitter",
"¿Sobre qué quieres buscar?")
   self.consultaTweets = self.dialogTweets.getInt(self,"Cuántos twits quieres
usar","La cantidad se multiplica por 100")
   self.nerCantidadValor = self.nerCantidad.getInt(self,"Cuántos twits quieres usar
en NER","Tweets que se usarán para NER")
   self.cargar_datos_de_twitter()
```

Esta función recoge tres valores importantes:

a. **La consulta:** Es decir, la búsqueda que se va a realizar en twitter para obtener la información, similar a la hora de buscar un solo tweet.

b. **Numero de páginas:** Cuando se quieren sacar datos de twitter, el servicio impone un límite de 100 tweets por cada consulta, es por eso que la técnica de la paginación consiste en llevar a cabo dos pasos para que se puedan sacar más de 100 tweets[30]:

    i. Realizar una iteración en un bucle por cada 100 tweets que queramos obtener.

[30] Si no puede sacar más de 100 tweets almacenará la cantidad de la que disponga y terminará el bucle con un aviso.

ii. Guardar el identificador del último tweet que se extrajo para continuar por ese tweet y no repetir descargas innecesarias.

Es por eso que uno de los datos que se le solicitan al usuario es la cantidad de paginaciones que se desean realizar. Realmente se le solicita el número de tweets que va a sacar de Twitter, pero el número que se indica es esa misma cantidad multiplicada por 100.

c. **Número de tweets utilizados en NER:** NER es un procedimiento costoso que ocupa bastante tiempo de CPU. Se quería evitar que el usuario solicitase descargar de Twitter 100.000 tweets y que analizase con NER esos 100.000 tweets, lo cual conllevaría un tiempo demasiado grande de espera. Es por ello que al usuario se le solicita el número de tweets que se van a utilizar para reconocer entidades. Esto no significa que el análisis de sentimientos se vaya a hacer solamente con los tweets utilizados para NER; se realizará el análisis de sentimientos para todos los tweets descargados en esa sesión, pero en base a las entidades obtenidas en los tweets escogidos.

2. Otro punto a tener en cuenta es que cuando se analiza un bloque de Tweets queremos mostrar los resultados por cada entidad reconocida, es por ello que al obtener los resultados de la función *usar_NER* lo que hacemos es realizar un análisis de aquellos tweets solamente que hablen de la entidad concreta que estamos analizando en cada caso.

```
resultadoNER = self.usar_NER(anNER,3)
for ent in resultadoNER:
    for it in ent:
        df2 = df2[df2['text'].str.contains(it[0])]
        df2 = self.tokenizar(df2)
        test_data = df2['final'][-100:]
        test_data = list(test_data.apply(' '.join))
        test_vectors = self.vectorizer.transform(test_data)
        self.mostrar_graph(self.predecir_Naive_Bayes(test_vectors,
it),self.predecir_SVC(test_vectors, it), self.predecir_KNN(test_vectors, it),
self.predecir_MLP(test_vectors, it))
        df2 = pd.DataFrame(data=df['text'])
```

3. El uso de la función *mostrar_graph* se detalla a continuación. Se trata de una función que, utilizando librerías de Python, muestra cuatro gráficas por cada entidad obtenida, cada gráfica corresponde al resultado del análisis de cada algoritmo.

```
def mostrar_graph(self, NB,SVC, KNN, MLP):
    #Naive Bayes
    plt.subplot(221)
    plt.title("NB para la entidad " + NB[0])
    plt.ylabel('tweets')
```

```
    plt.xticks(range(len(NB[1])), ['positive', 'neutral', 'negative'])
    plt.bar(range(len(NB[1])), height=NB[1], width = 0.75, align = 'center', alpha
= 0.8)
    #SVC
    plt.subplot(222)
    plt.title("SVC para la entidad " + SVC[0])
    plt.ylabel('tweets')
    plt.xticks(range(len(SVC[1])), ['positive', 'neutral', 'negative'])
    plt.bar(range(len(SVC[1])), height=SVC[1], width = 0.75, align = 'center',
alpha = 0.8)
    #KNN
    plt.subplot(223)
    plt.title("KNN para la entidad " + KNN[0])
    plt.ylabel('tweets')
    plt.xticks(range(len(KNN[1])), ['positive', 'neutral', 'negative'])
    plt.bar(range(len(KNN[1])), height=KNN[1], width = 0.75, align = 'center',
alpha = 0.8)
    #MLP
    plt.subplot(224)
    plt.title("MLP para la entidad " + MLP[0])
    plt.ylabel('tweets')
    plt.xticks(range(len(MLP[1])), ['positive', 'neutral', 'negative'])
    plt.bar(range(len(MLP[1])), height=MLP[1], width = 0.75, align = 'center',
alpha = 0.8)
    plt.show()
```

4. Así mismo, las diferentes funciones `self.predecir_Naive_Bayes`, `self.predecir_SVC`, `self.predecir_KNN` y `self.predecir_MLP` utilizadas en la llamada a *mostrar_graph*, son similares entre ellas y utilizan el método *predict* para predecir el sentimiento asociado en base al entrenamiento recibido. Veamos un ejemplo de una de ellas:

```
def predecir_KNN(self, test_vectors, it):
    mod = KNeighborsClassifier()
    file = open('Knn', 'rb')
    mod = load(file)
    result = mod.predict(test_vectors)
    pos = len(result[result == 4])
    neg = len(result[result == 0])
    neu = len(result[result == 2])
    y = [pos, neu, neg]
    return (it[0],y)
```

Funcionamiento de interfaz gráfica

El procedimiento gráfico para esta sección muestra la siguiente información:

En primer lugar, al igual que el resto de opciones, se muestra la información dentro de la ventana principal en el botón Analizar bloque desde Twitter.com

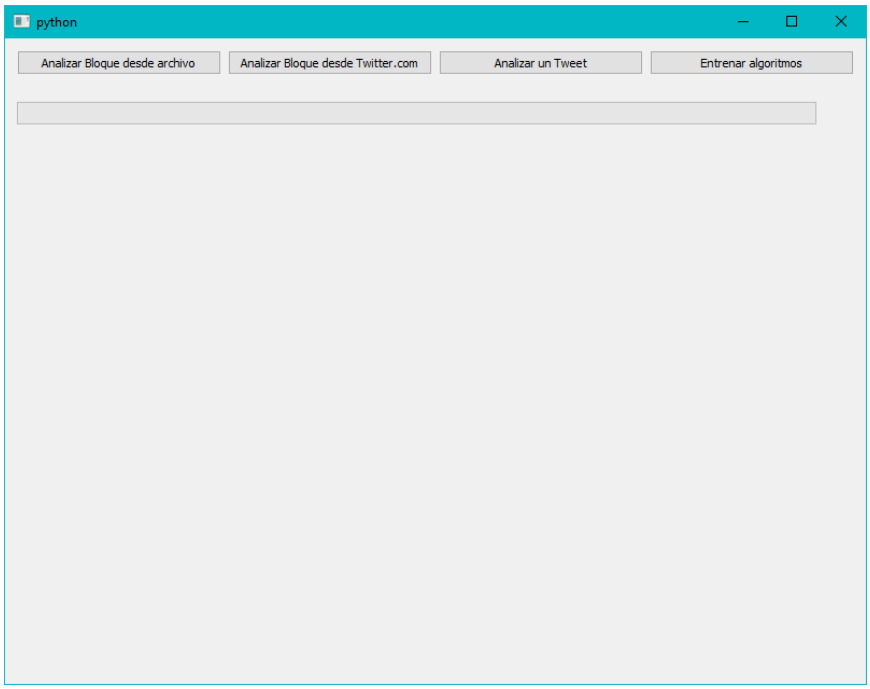


Ilustración 12: Ventana principal

Haciendo clic en este botón se abre una ventana que solicita, de la misma manera que en el caso de análisis de un tweet individual, el tema sobre el cual se va a buscar la información en twitter.

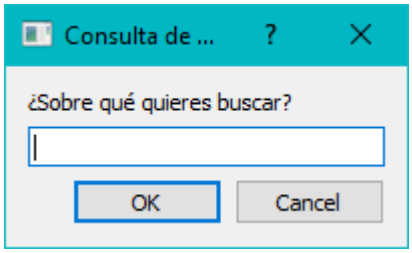


Ilustración 13: Ventana de selección del tema a buscar

A continuación, después de especificar el tema, aparecerá otra ventana que solicitará los datos referentes a la paginación.

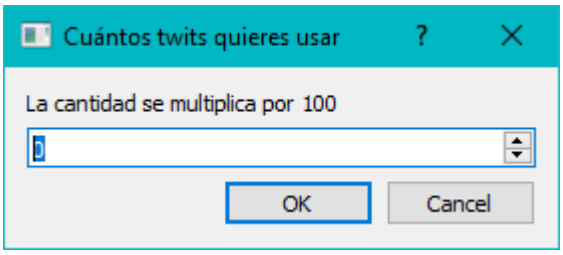


Ilustración 14: Ventana de selección de datos de paginación

La última información que se le solicita al usuario es la cantidad de tweets que van

a ser utilizados para el reconocimiento de entidades.

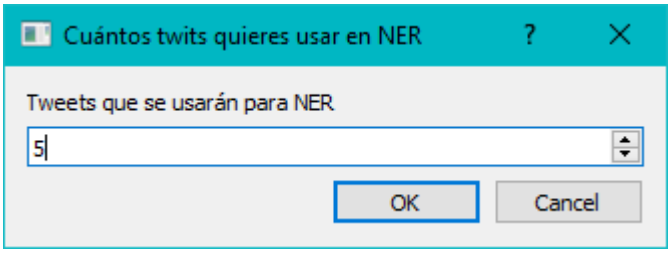


Ilustración 15: Ventana de selección de tweets a utilizar para NER

Por último, tras realizar el análisis correspondiente, tanto de sentimientos como de entidades, aparecerá una ventana emergente por cada entidad reconocida que mostrará cuatro gráficas correspondientes a los cuatro algoritmos utilizados para analizar sentimientos.

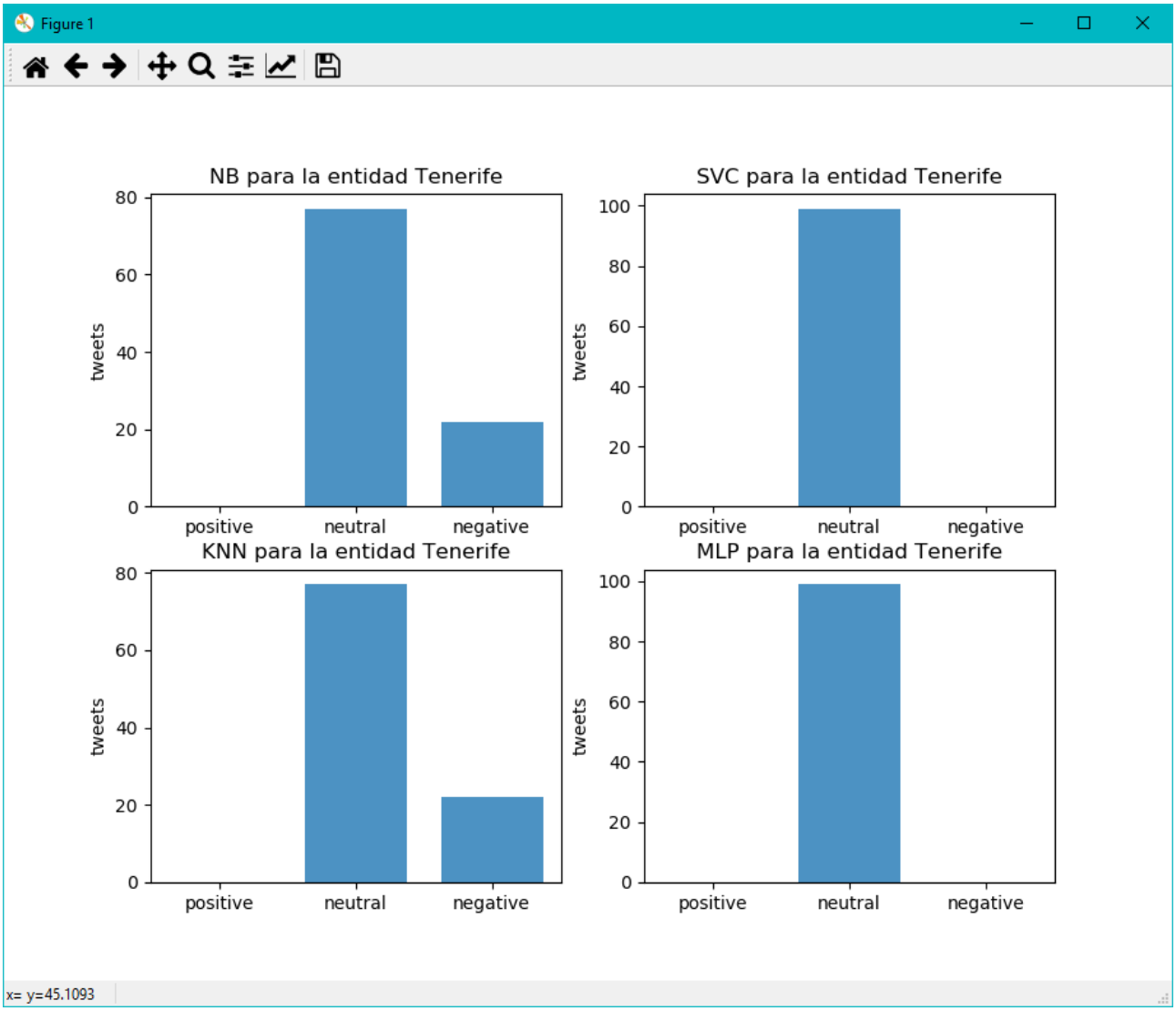


Ilustración 16: Ventana de resultados de análisis de un bloque de tweets de Twitter.com

4.6.4 Analizar un bloque de tweets desde un archivo

Pseudocódigo

Solicitar el archivo al usuario
Realizar el tokenizado de los datos
Realizar el reconocimiento de entidades con NER
Por cada entidad reconocida:
    Realizar el análisis de sentimientos por cada algoritmo
    Crear un gráfico que muestra los resultados

Detalles de la implementación

En este último caso sólo queda destacar que la función de analizar un bloque de tweets desde un archivo es muy similar a la función que analiza un bloque de tweets desde Twitter.com. La única diferencia se encuentra en el apartado de la interfaz gráfica y en que los datos no necesitan ser limpiados, sino solamente deben ser preparados para el clasificador.

Funcionamiento de interfaz gráfica

El funcionamiento de la interfaz gráfica en este caso es el siguiente:

Al igual que en el resto de funciones, si se quiere analizar un bloque de tweets simplemente hay que acceder a la ventana principal y seleccionar el botón Analizar Bloque desde archivo

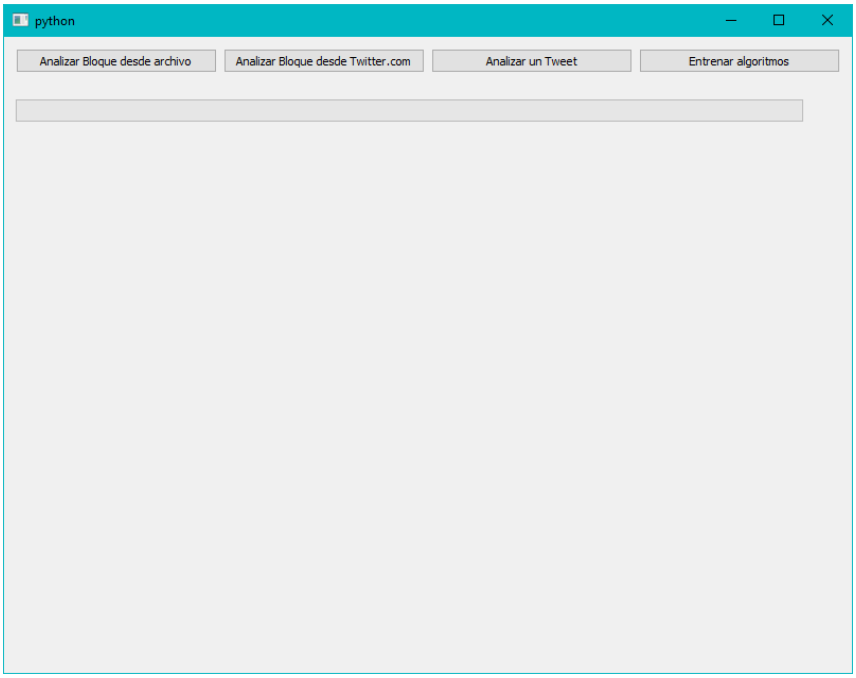


Ilustración 17: Ventana principal

A continuación, aparecerá un cuadro de diálogo solicitando al usuario que seleccione el archivo donde se encuentran los tweets que se quieren analizar.

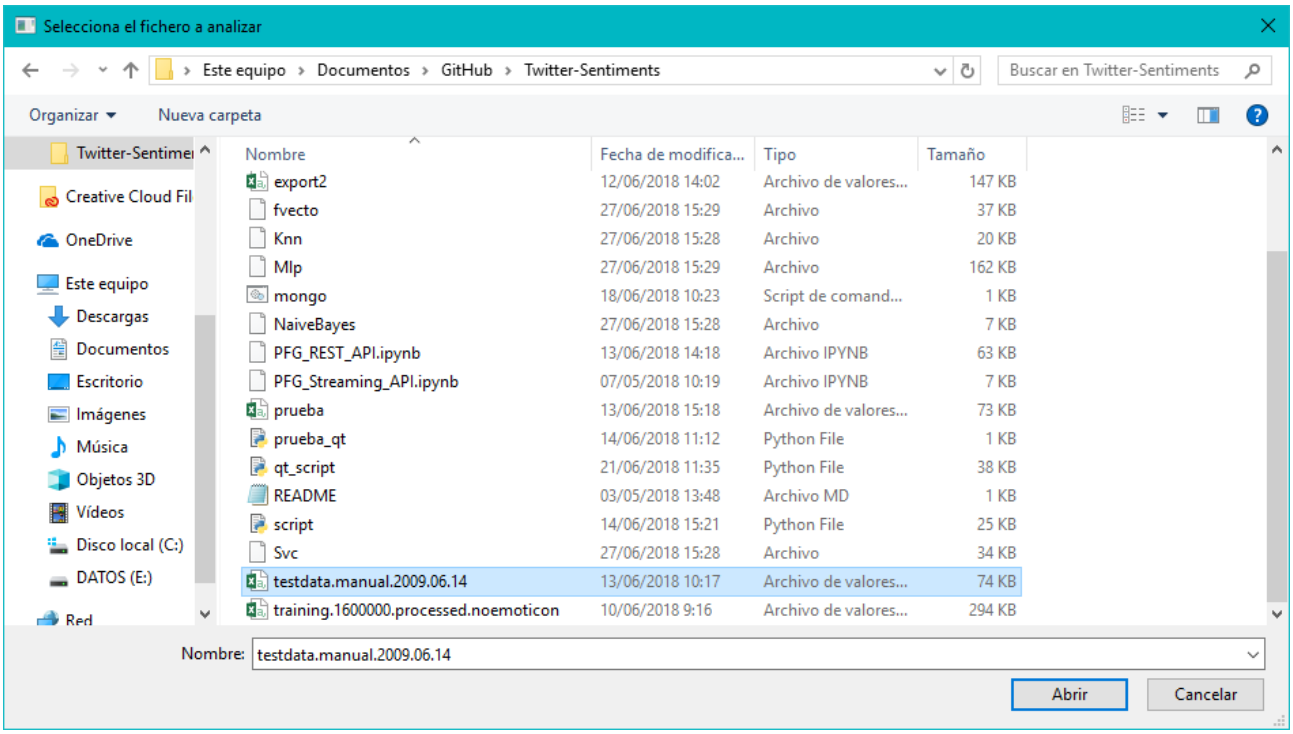


Ilustración 18: Ventana de elección de fichero a analizar

El siguiente paso es seleccionar cuántos de esos tweets van a ser usados para identificar entidades.

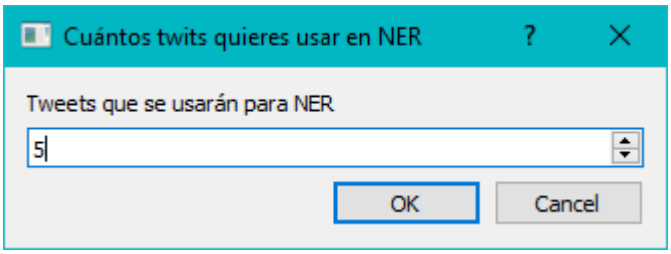


Ilustración 19: Ventana de elección de tweets para NER

Por último, después de hacer las operaciones necesarias, se desplegará una ventana donde aparecerán los resultados por cada entidad reconocida

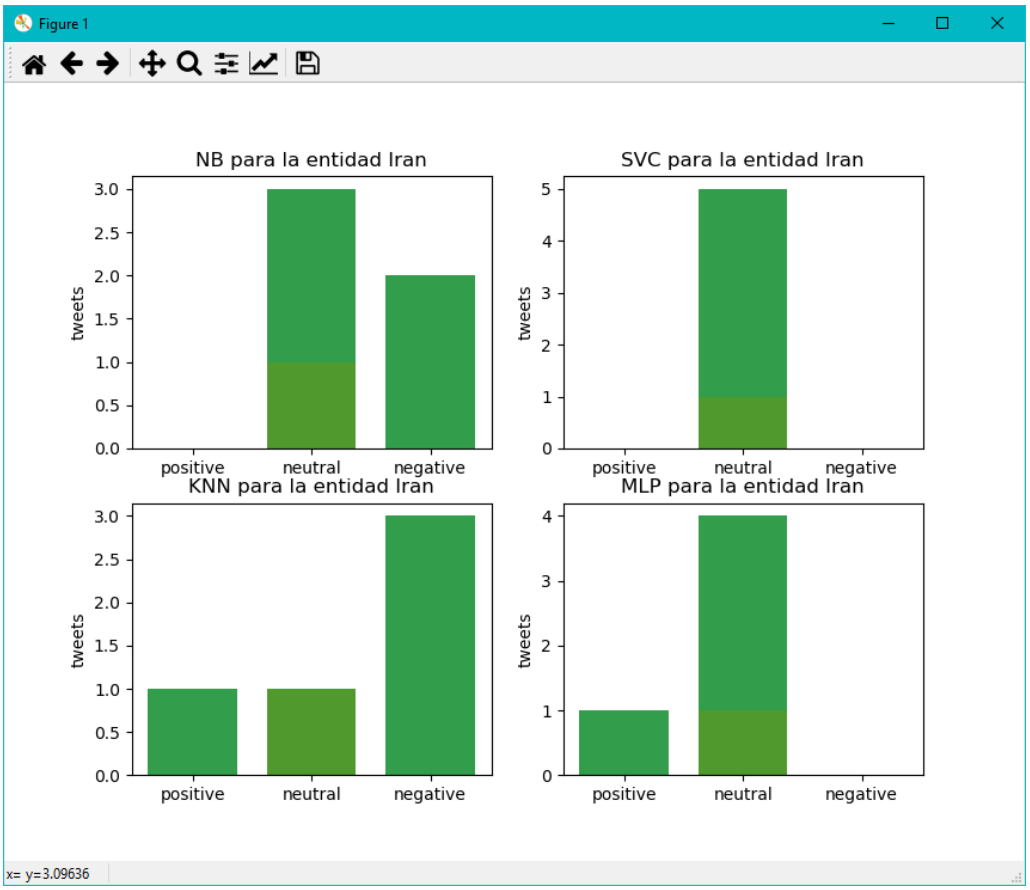


Ilustración 20: Ventana con los resultados del análisis desde un fichero

4.7 Pruebas

En el desarrollo de los diferentes elementos que conforman el proyecto, se realizaron diferentes tipos de pruebas.

4.7.1 Pruebas con diferentes datos.

Estas pruebas tenían la intención de comprobar el correcto funcionamiento de los procesos de limpieza y preparación de los datos para cualquier tipo de dato entrante. De esta manera se comprobó el funcionamiento de dichos procedimientos utilizando:

- Datos almacenados en un fichero .csv
- Datos almacenados en un fichero Excel.
- Datos obtenidos directamente desde twitter mediante REST API y Streaming API.

4.7.2 Pruebas en diferentes sistemas operativos.

Debido al desarrollo en múltiples puestos de trabajo, tuve que realizar el desarrollo tanto en macOS como en Windows 10. Si bien las diferencias entre ambos programas en los dos sistemas operativos eran mínimas, si requerían ciertas configuraciones o instalaciones de librerías que variaban de un sistema a otro.

En cualquier caso, mediante la instalación de las librerías necesarias y la correcta utilización y declaración de las variables de entorno, las pruebas resultaron satisfactorias para ambos sistemas operativos.

4.7.3 Pruebas con diferentes parámetros para los algoritmos.

Por último, y con motivo del objeto de este proyecto, se probaron distintas configuraciones de algoritmos para comprobar el correcto funcionamiento de los mismos y para realizar un breve estudio de los resultados obtenidos.

4.8 Problemas encontrados

Durante el desarrollo de este proyecto de fin de grado me he encontrado con distintos tipos de problemas que he ido solucionando con la ayuda de la tutora del proyecto, el cotutor y gracias a la extensa documentación que existe en internet.

Para detallar los problemas con los que me he encontrado, voy a dividirlos en tres subtipos generales: Conceptuales, IDE y Python y QT

4.8.1 Problemas conceptuales

Los problemas conceptuales fueron surgiendo a lo largo de todo el periodo de desarrollo del proyecto de fin de grado. Desde el comienzo tuve que enfrentarme a conceptos nuevos relacionados con clasificadores, redes neuronales, sistemas de aprendizaje, etc.

La mayoría de estos conceptos no los conocía tras mi paso por la carrera de Grado en Ingeniería Informática, [31]y aunque había oído bastantes veces conceptos como redes neuronales, no había tenido la oportunidad de estudiar el concepto como tal.

Y aunque, a fin de cuentas, el objetivo de este proyecto no es profundizar en el conocimiento de este tipo de conceptos sino más bien implementar los algoritmos y observar el comportamiento de los tales, vi necesario, no ahondar, sino tener una visión clara de qué significan estos conceptos.

Si bien no pude profundizar en cada algoritmo para tener un conocimiento extenso o experto de cada uno de ellos, si investigué la idea general para poder hacer uso de la parametrización de cada uno de ellos de una manera mínimamente lógica.

[31] Seguramente debido a que la única asignatura relacionada con este tema que cursé fue Inteligencia Artificial.

4.8.2 Problemas con el IDE y librerías de Python

Los mayores problemas que tuve en el proyecto sucedieron a la hora de utilizar las herramientas IDE y las librerías de Python. La recomendación de la tutora fue utilizar Jupyter Notebook y Anaconda Navigator para desarrollar el código. Casualmente, la versión de Anaconda Navigator que existía en el momento de iniciar el desarrollo presentaba múltiples fallos en Windows, así que tuve que comenzar el desarrollo únicamente en macOS.

Con la llegada de nuevas actualizaciones de la plataforma, la instalación y ejecución en Windows mejoraron y pude utilizar el IDE en ambos sistemas operativos.

En cuanto a las librerías de Python, tuve que utilizar las siguientes:

- **PySide2:** Se trata de una librería que implementa Qt for Python, imprescindible para la interfaz gráfica.
- **Pandas:** Esta librería se utiliza para integrar la estructura de datos *DataFrame* y utilizarla a lo largo del código para almacenar la información.
- **Requests:** Se utiliza para integrar funciones que realizan peticiones a páginas webs, en nuestro caso, a twitter.
- **Sklearn:** Esta extensa librería contiene las funciones necesarias para implementar los diferentes algoritmos de los que hacemos uso en el programa.
- **Pickle:** Utilizamos esta librería para poder guardar el estado de ciertas variables, concretamente las variables que almacenan los algoritmos entrenados.
- **Nltk:** Nltk se utiliza para implementar todo lo relacionado con NER y con el tratamiento del texto para tokenizar y detectar stopwords.

Durante el desarrollo del proyecto pude encontrarme con bastantes problemas que tenían que ver con instalaciones de librerías. Sin embargo, gracias a la sencillez con la que Anaconda Navigator es capaz de crear entornos y permite añadir librerías a dichos entornos, pude instalar, sin demasiados problemas, todas las librerías que necesitaba.

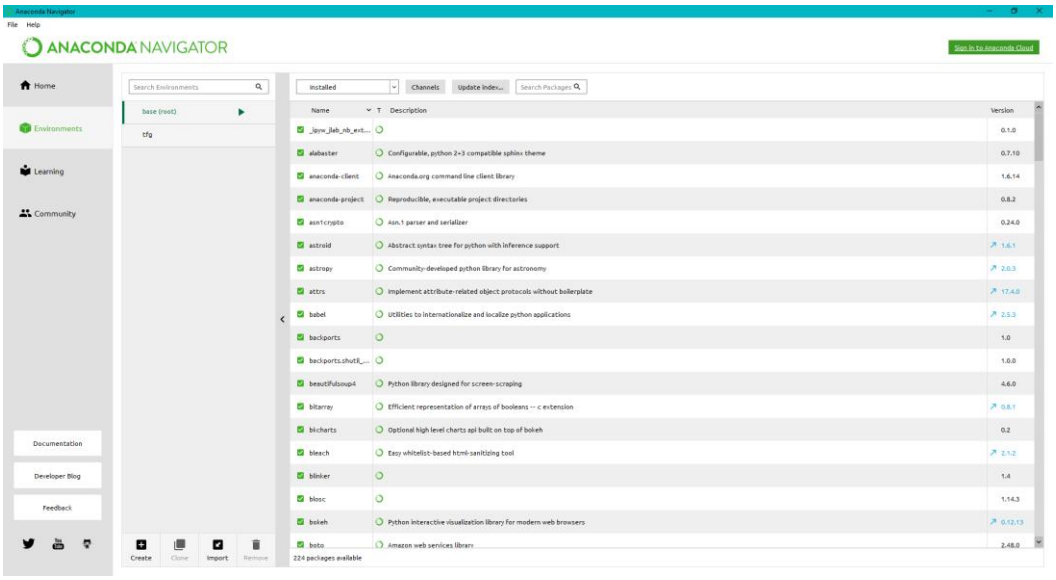


Ilustración 21: Entorno de trabajo de Anaconda Navigator

Como parte del proceso de desarrollo, además, creé un *cuaderno de bitácoras* donde apuntaba todos los problemas con los que me iba encontrando y el proceso llevado a cabo para solucionar cada uno de esos problemas. Para ello creé un blog de Google gratuito donde actualizaba toda la información hasta llegar al desarrollo en *Qt*[32]. Ya que el documento está pensado para ser visionado en un lector de PDF y no para ser impreso, el blog del que hago mención se puede consultar accediendo a este enlace.

4.8.3 Problemas con QT

En cuanto a QT los problemas que tenía se debían, en parte a dos causas: En primer lugar, no estaba familiarizado con el uso de QT, aunque había tenido algunas asignaturas en las que había desarrollado aplicaciones con QT con el cotutor, había sido hace ya bastante tiempo y había perdido la práctica, además, el uso que le había dado era en C++ y no en Python.

En segundo lugar, la documentación de PySide2 contenía muy pocos ejemplos, así que me resultaba bastante complicado averiguar y entender qué funcionamiento tenían las librerías, y aunque la nomenclatura utilizada por las mismas era bastante intuitiva, la falta de ejemplos me hizo consultar muchas dudas con el cotutor.

Con QT tuve 2 problemas generales:

- Entender el funcionamiento de la creación de ventanas y de Layouts.
- Entender e implementar el funcionamiento de los subprocesos dentro de QT.

[32] El desarrollo de QT no está detallado en este blog debido a que todos los conceptos eran nuevos, y resultaba una pérdida de tiempo documentar todo lo que aprendía o solucionaba.

4.9 Campos a mejorar

Todo proceso de desarrollo conlleva un margen de mejora una vez analizado el resultado. En el caso de este proyecto, cuando ya todas las funcionalidades operaban correctamente, pude darme cuenta de varias mejoras para hacer el código y la funcionalidad más eficiente de lo que es a la hora de entregar esta memoria:

1. Podría crearse un apartado, solicitando los datos al usuario, para que este pueda introducir las claves necesarias para la conexión a twitter según la cuenta que dese utilizar. Esto permitiría utilizar diferentes cuentas según el usuario que lo utilice. En el anexo de comentarios de futuros desarrollos veremos cómo hacerlo en código.
2. La variable *pms*[33], podría almacenarse en un fichero para que exista la posibilidad de guardar el estado de cada paginación según el tema que se ha buscado.
3. Podría existir la posibilidad de crear diferentes colecciones y tablas de bases de datos según la consulta que se vaya a realizar, esto conllevaría también a guardar a que tabla corresponde cada consulta por si se quiere acceder al historial completo de todas las consultas que se haya realizado con respecto a un tema.
4. Se podría indicar a la hora de solicitar el tema de la búsqueda cuales son los parámetros que se pueden utilizar en el campo de búsqueda de Twitter para que el usuario pueda utilizarlos en su búsqueda.
5. Se podría dar la oportunidad de configurar la búsqueda de tweets mediante los parámetros de configuración de twitter (lenguaje, tipo de resultado, etc)
6. Sería bueno poder utilizar una opción dentro del programa que permita seleccionar un algoritmo concreto para ser entrenado o para analizar un conjunto de datos sólo con un algoritmo.
7. Se podría solicitar al usuario que indique qué tipo de información se desea rechazar en la búsqueda de tweets.
8. Otra opción sería utilizar otros tipos de analizadores NER que encontrasen más de tres tipos de entidades.
9. También en cuanto a NER, se podría incluir en este análisis, la capacidad de detectar organizaciones, lugares o personas que no necesariamente tengan que comenzar con letra mayúscula, debido al lenguaje escrito actual, es cada vez más común ver este tipo de errores.
10. Se podrían etiquetar muchos más tweets para mejorar el entrenamiento de los algoritmos. Al mismo tiempo, se podrían conseguir tweets específicos

[33] Esta variable se utiliza en todos los procesos de obtención de tweets. Para más información, consultar el apartado *Extracción de tweets de twitter.com en Descripción de procesos.*

relacionados con el tema concreto.

11. La declaración de la variable *vectorizer*[34] actualmente es estándar, pero podría modificarse en pos de encontrar mejores resultados.
12. Se podría añadir al análisis en bloque un *widget* de tabla para mostrar todas las entidades que se han encontrado en el análisis, incluso, en vez de una tabla, podrían ser botones para mostrar el resultado de cada una de ellas.
13. Por otro lado, se podría añadir la opción de analizar el entrenamiento de los algoritmos para mostrar las mejores configuraciones, la validación cruzada resultante, las tablas de mediciones y la matriz de confusión.

[34] Vectorizer es una función que se encarga de establecer los parámetros que utilizarán los modelos. Para más información, se puede consultar las *variables importantes en Apéndice I.*

5. Conclusiones

Para poder abordar por completo las conclusiones obtenidas en este proyecto de fin de grado, me es necesario dividir este apartado en tres subapartados: Conclusiones en cuanto al análisis de sentimientos, conclusiones en cuanto a los algoritmos y conclusiones generales y líneas futuras.

5.1 Conclusiones en cuanto al análisis de sentimientos

En cuanto al análisis de sentimientos me gustaría recalcar varios puntos interesantes que he podido concluir durante el desarrollo de este proyecto:

- Existe una variedad bastante grande de sistemas, algoritmos y planteamientos en cuanto al análisis de sentimientos en las redes sociales. Evidentemente es un campo que cada vez se está convirtiendo más en imprescindible debido a que, en primer lugar, la minería de datos y el Big Data están proporcionando más información de la que podemos asumir y, en segundo lugar, las empresas y organizaciones necesitan poder aprovechar las innumerables opiniones que se expresan cada día en internet.
- A pesar de ello, la técnica es complicada, y conforme a la limitada experiencia que he podido adquirir durante el desarrollo de este proyecto puedo concluir que he encontrado algunas realidades que habría que tener en cuenta en cuanto al análisis de sentimientos:
    - **El lenguaje fluctúa con el tiempo:** Las expresiones cambian y, sobre todo, con la inclusión de la tecnología, la expresión escrita ya no depende solamente del idioma, sino de la cultura, la localización e incluso el momento temporal actual. La tecnología nos ha enseñado que lo que es, ya fue, y esto sucede exactamente igual con el lenguaje.
    - **El contexto es importante:** El análisis de sentimientos se basa única y exclusivamente en la información que se recoge del texto a analizar. Pero si se quiere obtener un resultado más fidedigno, seguramente el desarrollo de los algoritmos deba enfocarse en analizar también el contexto. Y el contexto en internet, por suerte o por desgracia, puede ser tan grande como se quiera.

- **El etiquetado:** Continuando con la idea del contexto, en el proceso de etiquetado de datos para el entrenamiento de algoritmos recibí la ayuda de personas distintas para que ellos decidieran si un tweet era positivo, negativo o neutro. Quizá un punto a tener en cuenta sea, no sólo elegir una muestra representativa de los tweets con los que se va a entrenar un algoritmo, sino también utilizar una muestra representativa de las personas que van a etiquetar dichos tweets.

5.2 Conclusiones en cuanto a los algoritmos

En el campo de los algoritmos, se ha tratado, en este proyecto, de comprobar el funcionamiento de los mismos. El proyecto, en un inicio, busca darle uso a un algoritmo de análisis de sentimientos con la temática del turismo en la isla de Tenerife y analizar los resultados, pero tras tener la primera reunión con la tutora, se cambió a los carnavales de Tenerife.

A pesar de que se condujo el proyecto por esa línea, los tweets obtenidos para entrenar los algoritmos en ese ámbito eran muy pocos, por lo tanto, el entrenamiento no iba a ser fiable. Es por ello que se buscó una manera de reestablecer el objetivo del proyecto: No se utilizaría un solo algoritmo, sino varios, y el objetivo no sería estudiar los sentimientos correspondientes a ese tema, sino, más bien, elaborar una aplicación con la cual se pudiera estudiar cómo se comportan diferentes algoritmos con el mismo entorno de entrenamiento con una búsqueda cualquiera.

Los resultados, en este sentido, dependen de la calidad del entrenamiento realizado, así como de los parámetros escogidos. En referencia a los parámetros, se realiza una búsqueda de la mejor opción. En cuanto a la calidad del entrenamiento, se pudieron etiquetar 1100 tweets. Además de que puedan resultar pocos tweets, estos tweets eran de un contexto generalista, y no específico, que era lo que se buscaba en un inicio, por lo tanto, no es la mejor manera de entrenar los algoritmos.

A continuación, voy a explicar las conclusiones obtenidas de cada uno de ellos:

SVC

Configuración: [Kernel=Linear, Degree=3, coef0=0,1, Probabilidad=True, Shrinking = True]

Validación Cruzada: 0.456

Mediciones:

SVC	Precisión	Memoria	Puntuación F-1	Soporte
0	0.48	0.27	0.35	44
2	0.22	0.91	0.36	22
4	0.73	0.14	0.23	59
Media/Total	0.55	0.32	0.29	125

Tabla 4: Mediciones de SVC

Matriz de confusión:

Predichos/Reales	**0**	**2**	**4**
0	12	29	3
2	2	20	0
4	11	40	8

Tabla 5: Matriz de confusión de SVC

MLP

Configuración: [Tamaño de capas ocultas = 150, Activación = Tanh, Alpha = 0.001, Aprendizaje = Constante]

Validación Cruzada: 0.416

Mediciones:

MLP	Precisión	Memoria	Puntuación F-1	Soporte
0	0.54	0.3	0.38	44
2	0.18	0.55	0.27	22
4	0.47	0.27	0.34	59
Media/Total	0.44	0.33	0.34	125

Tabla 6: Mediciones en MLP

Matriz de confusión:

Predichos/Reales	**0**	**2**	**4**
0	13	21	10
2	2	12	8
4	9	34	16

Tabla 7: Matriz de confusión de MLP

KNN

Configuración: [Número de vecinos = 1, Pesos = uniformes, algoritmo = ball_tree, leaf_size = 5, p = 1]

Validación Cruzada: 0.424

Mediciones:

KNN	Precisión	Memoria	Puntuación F-1	Soporte
0	0.38	0.56	0.44	44
2	0.15	0.18	0.16	22
4	0.47	0.27	0.34	59
Media/Total	0.38	0.35	0.35	125

Tabla 8: Mediciones de KNN

Matriz de confusión:

Predichos/Reales	0	2	4
0	24	9	11
2	11	4	7
4	29	14	16

Tabla 9: Matriz de confusión de KNN

Si comparamos los algoritmos obtenemos la siguiente información:

Clasificador	Precisión	Memoria	Puntuación F-1	Validación Cruzada
SVC	0.55	0.32	0.29	0.456
MLP	0.44	0.33	0.34	0.416
KNN	0.38	0.35	0.35	0.424

Tabla 10: Comparativa de clasificadores

Por lo tanto, podemos concluir que para el tipo y tamaño de muestras, y las pruebas realizadas, SVC, con la configuración encontrada, es el mejor clasificador que podríamos usar, tengamos en cuenta que KNN presenta una mejor puntuación F-1 y una mejor memoria, sin embargo, la precisión de SVC (incluso superando el 50% con tan pocos datos) y su validación cruzada, son mayores, lo que hace que en definitiva se convierta en el mejor clasificador para este caso concreto.

Por otro lado, los resultados distan mucho de ser los deseados, si bien por la

cantidad de datos utilizados o por el ámbito de la búsqueda los resultados son un más bajos de los útiles para realizar este tipo de análisis (entre el 75 y 85% de precisión) Así que, evidentemente, se concluye que tan importante como los algoritmos de análisis son las muestras de datos para realizarlos.

5.3 Conclusiones y líneas futuras

Internet se ha convertido en el lugar donde el ser humano se expresa. No sólo donde almacena información relativa a su trabajo o donde puede ver vídeos o fotografías, sino que con el tiempo se ha demostrado que lo más importante de la red de redes es la información, y de esta información, la opinión de las personas es la más valiosa.

Con este proyecto se ha demostrado la capacidad actual de los sistemas de análisis de sentimientos en una plataforma tan importante para la opinión como es Twitter.com. Mediante la implementación de un aplicativo para ordenadores de sobremesa es muy sencillo acceder a datos relativos a la opinión de millones de personas sobre un tema en particular. Como un trabajo o desarrollo futuro se podrían implementar nuevos algoritmos que se adaptasen al ámbito que se desea estudiar, así como la utilización de estas técnicas en otras redes sociales, incluso, la implementación de un algoritmo clasificador propio.

Evidentemente, el campo del análisis de sentimientos *está en pañales*. A medida que la tecnología avance y se encuentren nuevas técnicas para para la minería de datos y el tratamiento de Big Data, estoy convencido de que podremos ver cambios en internet y, sobre todo, en las redes sociales que no dejarán de asombrarnos. Los datos están ahí. Sólo hay que saber usarlos.

5.4 Conclusions and future directions

The Internet has become the place where human beings express themselves. Not only where you store information about your work or where you can watch videos or images, but over time it has been shown that the most important thing about the network of networks is information, and from this information, people's opinions are the most valuable.

With this project, the current capacity of the feeling analysis systems has been demonstrated on such an important platform for opinion as Twitter.com. By implementing a desktop application it’s very easy to access data on the opinions of millions of people about a specific topic. As a future work or development, new algorithms could be implemented that adapt to the scope to be studied, and the use of these techniques in other social networks, including the implementation of an own sorting algorithm.

Obviously, the field of sentiment analysis is young. As technology advances and new techniques are found for data mining and Big Data processing, I am convinced that we will see changes on the Internet and, above all, on social networks that will not cease to amaze us. The data is there. We just have to know how to use them.

6. Bibliografía

Araujo, B. S. (2010). *Aprendizaje automático: un enfoque práctico.* Madrid: Ra-Ma.

Chatterjee, S., & Krystyanczuk, M. (2017). *Python Social Media Analytics.* Birmigham: Packt Publishing.

cjhutto. (s.f.). *Repositorio VADER*. Obtenido de GitHub: https://github.com/cjhutto/vaderSentiment

García, L. M. (2014). *Análisis de sentimientos y predicción de eventos en twitter.* Santiago de Chile.

Matich, D. J. (2001). *Redes Neuronales: Conceptos básicos y aplicaciones.* Rosario.

MongoDB. (s.f.). *Documentación Mongo*. Obtenido de MongoDB: https://docs.mongodb.com/

Moreno, A., Armengol, E., Béjar, J., Belanche, L., Cortés, U., Gavaldá, R., . . . Sánchez, M. (s.f.). *Aprendizaje automático.* Eidicions UPC.

Moreno, I. (s.f.). *Reconocimiento y Clasificación de Entidades Nombradas independiente de la lengua y el dominio mediante perfiles.* Alicante: Universidad de Alicante.

Parra, R. A. (2016). *Redes Neuronales Parte 1.* Universidad Libre.

PySide2. (s.f.). *Documentación QT for Python*. Obtenido de Qt.io: https://doc-snapshots.qt.io/qtforpython/index.html

SciKit Learn. (s.f.). *Documentación Clasificador KNN*. Obtenido de scikit-learn.org: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

SciKit Learn. (s.f.). *Documentación Clasificador MLP*. Obtenido de scikit-learn.org: http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

SciKit Learn. (s.f.). *Guía de usuario de SVC*. Obtenido de scikit-learn.org: http://scikit-learn.org/stable/modules/svm.html#svm-classification

SciKit Learn. (s.f.). *Guía de usuario para KNN*. Obtenido de scikit-learn.org: http://scikit-learn.org/stable/modules/neighbors.html

Scikit-Learn. (s.f.). *Documentación algoritmo SVC*. Obtenido de scikit-learn.org: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Wikipedia. (s.f.). *Análisis de sentimiento*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/An%C3%A1lisis_de_sentimiento

Wikipedia. (s.f.). *Big Data*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Macrodatos

Wikipedia. (s.f.). *Matriz de confusión*. Obtenido de Wikipedia.org: https://es.wikipedia.org/wiki/Matriz_de_confusi%C3%B3n

Wikipedia. (s.f.). *Programación Extrema*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema

Wikipedia. (s.f.). *Twitter*. Obtenido de Wikipedia.org: https://es.wikipedia.org/wiki/Twitter

Apéndice I: Manual del Programador

Con el fin de proporcionar una guía rápida y sencilla para futuras modificaciones del programa desarrollado en este proyecto, a continuación pasa a detallarse un manual donde se explican las principales características a tener en cuenta mediante un listado de la declaración de las funciones y variables incluyendo el propósito de su utilización.

Clases utilizadas

En este proyecto, al tratarse de un aplicativo sencillo, sólo se ha necesitado utilizar una clase. La clase en cuestión se llama Ventana, y es utilizada principalmente para realizar toda la funcionalidad establecida por QT for Python y encapsular el código de manera mínimamente ordenada.

Variables importantes

Dentro del programa existen ciertas variables que debemos de tener en cuenta de cara a futuras modificaciones:

- **Vectorizer:** Esta variable, declarada en la función principal del programa, se encarga de almacenar el resultado de la función de scikit learn, TfidfVectorizer. Esta función se encarga de convertir una colección de documentos sin procesar en una matriz de funciones TF-IDF, dicho de otro modo, y para que se entienda, establece las características básicas que utilizarán todos los modelos. Estas características son las siguientes:
    - Min_df: Descarta las palabras que aparezcan en menos tweets de los que se especifiquen en este parámetro.
    - Max_df: Descarta las palabras que aparezcan en más tweets de los que se especifiquen en este parámetro (es un porcentaje)
    - Sublinear_tf: Usa ponderación sublinear (escala el término frecuencia en escala logarítmica)
    - Use_idf: Activa la frecuencia de documento inversa.

  También es importante tener en cuenta que esta variable se almacena en un fichero para actualizar los cambios, además de que se le pasa al constructor de la clase ventana y se iguala a la variable self.vectorizer para ser utilizada en toda la clase.

- **Variables de interfaz:** Estas variables son las utilizadas para mostrar los elementos de la interfaz. Se declaran en el constructor de la clase Ventana.
    - **Variables de Layout:** Los Layouts son *cajones* donde se pueden

guardar todo tipo de widgets para ser mostrados. Existen tres tipos, pero nosotros utilizaremos dos: QHBoxLayout y [35]QVBoxLayout En nuestra clase tenemos los siguientes:

- **layoutMenu:** (QHBoxLayout) Se trata del layout que encierra todos los elementos que se encuentran en el menú, en nuestro caso, los cuatro botones principales.
- **layoutWidget:** (QVBoxLayout) Este layout se utiliza para ir mostrando la información necesaria según cada funcionalidad. Principalmente se utiliza en la función de analizar un solo tweet.
- **infoLayout:** (QHBoxLayout) Es el layout encargado de mostrar el estado actual del proceso de cada función. En otras palabras, en él se coloca un Label que muestra un texto de lo que el programa está haciendo durante los procesos.
- **progressLayout:** (QHBoxLayout) Es un layout donde se coloca una barra de progreso que muestra el porcentaje restante para la terminación de los procesos que realiza el programa.
- **layoutPrincipal:** (QVBoxLayout) Este layout es el que contiene al resto de layouts.

- **Variables de botones:** Sólo existen 4 variables para los botones (que se declaran con el método QPushButton) y todas están en el layoutMenu.
  - **buttonBloqueFile:** Al ser pulsado, activa la función analizar un bloque de tweets desde un archivo
  - **buttonBloqueTwitter:** Al ser pulsado, activa la función analizar un bloque de tweets desde Twitter.com
  - **buttonUnTweet:** Al ser pulsado, activa la función analizar un solo tweet obtenido de Twitter.com
  - **buttonEntrenar:** Al ser pulsado, activa la función de entrenamiento de los algoritmos
- **Variables de etiquetas:** Estas variables muestran información que queremos revelar a lo largo del programa, se declaran mediante el método QLabel.
  - **tituloLabel:** Muestra el título de la función que se está ejecutando en cada momento.

[35] QHBoxLayout organiza los widgets u otras layouts de manera horizontal, mientras que QVBoxLayout lo hace de manera vertical.

- **tweetLabel:** Muestra, en la función de analizar un solo tweet, el contenido de ese tweet.
- **nerLabel:** Muestra el título del análisis de entidades en la función de analizar un solo tweet.
- **nonerLabel:** Muestra una alerta cuando en el análisis de un solo tweet no se encuentra ninguna entidad.
- **claLabel:** Muestra un título para el análisis de sentimientos en la función de analizar un solo tweet
- **entrenamientoLabel:** Muestra un título para la función de entrenar algoritmos.
- **progressLabel:** Muestra los procesos que van realizándose en cualquier proceso del programa.

- **Variables de tablas:** Para poder mostrar datos de manera eficiente, en el análisis de un solo tweets, se crean dos tablas que muestran el resultado del análisis de sentimientos y el resultado del análisis de NER. Esta funcionalidad se basa en la clase QTableWidget de Pyside2[36]
  - **tabla:** Esta tabla muestra cada uno de las entidades encontradas
  - **tablaSent:** Esta tabla muestra el resultado por cada algoritmo.
- **Variables de cuadros de diálogo:** Para poder pedir información al usuario se han creado tres variables de cuadros de diálogo a través de la clase QInputDialog.
  - **dialogConsulta:** Solicita los datos de búsqueda en twitter.
  - **dialogTweets:** Solicita la cantidad de páginas que se van a utilizar en la paginación, o lo que es lo mismo la cantidad de tweets a buscar multiplicados por 100.
  - **nerCantidad:** Solicita la cantidad de tweets que se van a utilizar para detectar entidades con NER.

Además, dentro de esta clase existen diversas funciones que serán explicadas a continuación:

Funciones de la clase Ventana:

`def __init__(self, vectorizer, parent=None):` El constructor de la clase recibe como único parámetro relevante la variable vectorizer. La función del constructor está diseñada para suplir dos necesidades: Declarar las variables

[36] Al igual que el resto de clases para crear toda la interfaz gráfica.

necesarias para el funcionamiento del programa y conectar los botones a las diferentes funciones.

`def limpiarLayout(self,layout):` Esta función se encarga de limpiar el layout que se le pase como parámetro. Su funcionamiento es sencillamente un for que recorre el layout indicado y oculta todos los widgets. Es muy útil para cambiar entre las funciones que realiza el programa. ¿Por qué lo ocultamos y no los eliminamos? Porque crear widgets es costoso, y no tiene sentido destruirlos y crearlos en cada momento, cuando podemos ocultarlos y cambiar su contenido cuando queramos. Tiene una función contraria llamada `def mostrarLayout (self,layout)` que se encarga de mostrar todos los widgets que hay en un layout.

`def cuadroDialogo(self):` La función de cuadro de diálogo se encarga de mostrar los cuadros de diálogos correspondientes a analizar un solo tweet. La manera en que lo hace es creando tres variables y asignándole el resultado de las variables de cuadros de diálogo correspondientes asignadas en el constructor. Estas variables serán utilizadas a lo largo de diversas funciones, por lo tanto, son variables de clase.

`def cargar_datos_de_twitter(self):` Esta función se encarga de realizar todo el procedimiento para cargar y analizar los datos obtenidos desde Twitter.com. Dentro de esta función debemos tener en cuenta los siguientes aspectos:

- Hace uso de las variables consumer_key, consumer_secret, access_token y access_token_secret. Estas variables son las credenciales necesarias para utilizar la API de Twitter y así poder descargar la información que necesitemos. Si se quiere utilizar la API de Twitter se deberá utilizar el procedimiento que se explica en https://apps.Twitter.com/[37]
- La variable q se utiliza para realizar la consulta. Es importante tener en cuenta que si queremos modificar la búsqueda debemos alterar tanto el contenido de string directo de la variable como la variable `self.consultaText[0]` que recoge el resultado de la consulta introducida por el usuario.
- La variable pms establece los parámetros de la consulta explicados en la memoria, es necesario tener en cuenta que estos parámetros vienen establecidos igualmente por la API de Twitter.

[37] Si se quiere más información, se puede consultar (Chatterjee & Krystyanczuk, 2017) Python Social Media Analytics, concretamente la página 100, en la sección Getting Twitter API Keys.

- En cuanto a la base de datos, tenemos 3 variables que se pueden modificar:
    - Database_name: que guarda el nombre de la base de datos que queremos crear o utilizar.
    - Collection_name: que almacena el nombre de la colección que estará dentro de la base de datos antes nombrada.
    - Client: Esta variable es de tipo MongoClient y guarda la dirección a donde se conectará el programa. Se trata de un string con el siguiente formato: 'mongodb:<dirección>' con un ejemplo se entiende mejor: 'mongodb://localhost:27017/'. Este ejemplo es el lugar por defecto donde se ejecuta mongo, en local. Para acceder de una forma visual y sencilla a esta ejecución de mongo recomiendo usar MongoDB Compass Community. Una aplicación que permite acceder a las bases de datos de mongo, visualizar su contenido, editarlo y exportarlo a diferentes formatos.

`def analizarDeArchivo(self):` La función siguiente es la función que se encarga de recoger datos de un archivo csv y analizar los tweets que contenga. En esta función es importante que tengamos en cuenta los siguientes puntos:

- El fichero a analizar debe ser un fichero csv, debido a que la función que utilizamos es `pd.read_csv` de pandas. La librería Pandas incluye otro tipo de funciones para poder sacar datos de ficheros utilizando el formato Excel. Si se necesita utilizar este formato simplemente habría que cambiar esta función.
- También en cuanto a la estructura de dicho archivo, hay que tener en cuenta que la primera fila del archivo contiene las cabeceras de las columnas. En nuestro caso la cabecera que importa se llama text. Sería la columna donde se almacenarán los tweets.

`def analizarUnTweet(self):` Esta función lleva a cabo los procedimientos necesario para obtener un tweet de Twitter y analizarlo. La información acerca de como configurar la consulta se encuentra en la descripción de la función `def cargar_datos_de_twitter(self)` de este mismo anexo. Sin embargo, hay que tener en cuenta lo siguiente:

- Al igual que para el resto de funciones, pero en especial para esta, ya que es la que más uso le da a la interfaz gráfica debemos tener en cuenta el uso que se le da a `def limpiarLayout(self,layout)` y a `def mostrarLayout (self,layout).` Es importante que cada vez que se

vaya a hacer uso de esta función se utilicen las dos nombradas para que la interfaz gráfica funcione correctamente.

- A la hora de mostrar el tweet por la interfaz gráfica debemos tener en cuenta el formato de salida. Se debe extraer de nuevo el contenido del tweet, sin realizar las limpiezas, eliminación de *stopwords* y *tokenizar* para mostrar el tweet tal como se ha escrito, sin el formato necesario para realizar los análisis.

`def tokenizar(self, df):` La función tokenizar se encarga de realizar el proceso de tokenización (explicado con anterioridad) para cualquier dataframe pasado a la función. Como resultado de la tokenización, se devuelve el mismo dataframe tratado. Es importante para esta función tener en cuenta que hay que añadir las librerías para que las funciones `TweetTokenizer().tokenize`, `stopwords.words('english')` y `string.digits` funcionen correctamente. Esas funciones son: `from nltk.tokenize import TweetTokenizer`, `from nltk.corpus import stopwords` y `strings`.

`def limpieza_de_datos_de_twitter(self, df):` Al igual que tokenizar, esta función recibe un dataframe y elimina las fuentes del tweet que no queremos utilizar. Si se quiere agregar cualquier tipo de filtro o condición extra, es necesario añadirlo en esta función.

`def predecir_Naive_Bayes(self, test_vectors):` Esta función, y las que comparten un nombre similar, como predecir_SVC, predecir_KNN y predecir_MLP tienen como objeto realizar, con el modelo correspondiente, la predicción de los valores que reciba en la variable test_vectors. Los pasos que se llevan a cabo en estas funciones son:

1. Crear una variable para el modelo correspondiente.
2. Abrir el archivo que almacena el modelo entrenado previamente.
3. Realizar la predicción.
4. Almacenar en tres variables (pos, neg y neu) las cantidades de valores correspondientes a cada sentimiento.
5. Crear un vector con dichos valores.
6. Devolver dichos valores.

`def mostrar_graph(self, NB,SVC, KNN, MLP):` La función *mostrar_graph* se encarga de mostrar los gráficos correspondientes a cada modelo. Es por eso que en su declaración recibe los cuatro modelos usados y, a continuación, utilizando la librería *matplotlib.pyplot* crea un diagrama de barras para cada uno de ellos.

`def usar_NER(self, tweetys, n):` Esta funcionalidad permite realizar el análisis NER para obtener las entidades correspondientes por cada tweet que se le pase. Tiene dos parámetros.

- **Tweetys:** Es un listado de los tweets que se van a analizar. Este listado se recorre en un bucle *for* y se analiza, almacenando los resultados en una variable previamente creada, llamada *entities*.
- **N:** Esta variable es por si se quieren realizar modificaciones en el código. Lo que hace es determinar cuántas, de las entidades reconocidas, se van a utilizar. Para entenderlo mejor; Si un usuario solicita reconocer las entidades en 100 tweets, puede ser que reconozca 10 entidades, 50 o más de 300. Para evitar que el número de entidades reconocidas suponga un problema de ejecución o visualización, establecemos una variable que determine, de las entidades más nombradas, cuántas se van a usar. Si encontramos que se usan *[Obama=203, Atlanta=45, Lebron=41, Murcia=23,…]* y establecemos *n=2* obtendremos como resultado simplemente las entidades de Obama y Atlanta. Esta cantidad depende de la utilidad que se le vaya a dar al programa. En este caso, para analizar un bloque de tweets utilizamos las 3 entidades más nombradas y para analizar un tweet solamente utilizamos las 10 más nombradas.[38]

Por último, es importante aclarar que la función puede acabar de dos maneras, Si no encuentra ninguna entidad devuelve un 0, para informar de que no se ha encontrado ninguna entidad, sobre todo a la función de analizar un solo tweet. Si encuentra alguna entidad devuelve un vector con las entidades encontradas.

`def entrenar_algoritmos(self):` La función entrenar algoritmos no debe confundirse con la función *entrenar*. Esta función se encarga de realizar todos los procedimientos previos al entrenamiento, mientras que la función *entrenar* se encarga del entrenamiento en sí mismo. Debemos tener en cuenta los siguientes aspectos:

- Esta función, al igual que la que analiza un bloque de tweets desde un archivo, obtiene los datos de ficheros. En esta ocasión, de dos ficheros; uno para los datos de prueba y otros para los datos de entrenamiento. Igual que con la función recién citada, tenemos que tener en cuenta que estos ficheros deben estar en formato csv.

[38] Utilizamos 10, porque suponemos que en un tweet no podrían existir más de 10 entidades.

- La variable *puntuaciones* se encarga de almacenar las mejores puntuaciones de cada algoritmo y mantiene el mismo orden que la variable *modelos*. Si se quieren añadir nuevos modelos, habría que modificar ambas variables.
- Las variables *best_svc, best_knn, best_mlp*, almacenan la configuración de parámetros de los algoritmos con mejores resultados. En los sucesivos bucles *for,* al encontrar un resultado mejor que el actual, esta variable se modifica y guarda todos los parámetros que necesita.
- *Best_model* es una variable que se usa para almacenar el mejor modelo de todos. En la versión actual del programa no se utiliza, pero podría ser utilizado en posteriores versiones para realizar análisis simplemente utilizando el mejor modelo posible.

`def entrenar(self, alg, train_vectors, train_labels, test_vectors, test_labels, mod) :` Por último, la función entrenar se encarga de entrenar un modelo pasado por parámetros con unos vectores de entrenamiento y pruebas también pasados por parámetros. Se debe tener en cuenta que esta función, además de entrenar el modelo, lo almacena en un fichero para que no sea necesario entrenar el modelo cada vez que se ejecute el programa. Las variables a considerar son:

- Alg: Almacena el nombre del modelo que se va a entrenar a continuación.
- Mod: Almacena la variable con el modelo que se va a utilizar.

La función devuelve el resultado del entrenamiento.