



Universidad Autónoma del Estado de
México
Centro Universitario UAEM Zumpango

INGENIERÍA EN COMPUTACIÓN

ANÁLISIS DE SENTIMIENTOS DE TEXTOS EN
ESPAÑOL SOBRE COVID19 EN TWITTER

TESIS

Que para obtener el título de Ingeniera en Computación

PRESENTA

Esteban Lira Alanis

Director de Tesis
Dr. Asdrúbal López Chau

Co-Director de Tesis
Dr. Valentín Trujillo Mora

Zumpango, Estado de México. Mayo 2025



Contenido

1	INTRODUCCIÓN	1
1.1	Planteamiento del problema	2
1.2	Objetivos	2
1.2.1	Objetivo general	2
1.2.2	Objetivos específicos	2
1.3	Hipótesis	3
1.4	Justificación	3
1.5	Alcance	3
2	MARCO TEÓRICO	5
2.1	COVID19	5
2.2	Análisis de sentimientos	7
2.2.1	Métodos para análisis de sentimientos	9
2.2.2	Proceso general para realizar análisis de sentimientos	12
2.2.3	Bibliotecas para análisis de sentimientos	19
2.3	JSON	21
2.4	Lenguaje de programación Python	22
2.4.1	Cadenas en Python	23
2.4.2	Listas en Python	26
2.4.3	Estructuras de control de flujo en Python	30
2.4.4	Excepciones	37
2.5	UML	40
2.5.1	Diagrama de clases	41
2.5.2	Diagrama de casos de uso	43
2.5.3	Diagrama de actividades	44
2.6	Frecuencia de término - Frecuencia inversa de documento (TF-IDF) . .	46
2.7	Máquinas de Vectores Soporte	48
3	DESARROLLO	52
3.1	Proceso general para el análisis de sentimientos	52
3.1.1	Diagramas UML de implementación	54
3.2	Descripción detallada de los procesos	58
3.2.1	Recolección de publicaciones en Twitter	58

3.2.2	Proceso de descarga de tweets	60
3.3	Clasificación o etiquetado del conjunto de datos	67
3.4	Pre-procesamiento de los datos	68
3.4.1	Diagrama UML de clase detallado de la arquitectura	79
3.5	Proceso de análisis y entrenamiento	81
3.5.1	Análisis de sentimientos	81
3.5.2	Entrenamiento	87
4	PRUEBAS REALIZADAS	94
5	CONCLUSIONES	99
	Referencias	101

Lista de figuras

2.1	Cadenas de texto en Python.	24
2.2	Diagrama de flujo de la sentencia for.	31
2.3	Diagrama de flujo de la sentencia while.	33
2.4	Autómata simple para identificar números binarios pares.	36
2.5	Relaciones UML en diagrama de clases.	42
2.6	Relaciones UML en diagrama de casos de uso.	44
2.7	Documentos de ejemplo.	47
2.8	TF-IDF completo para los 3 documentos.	48
2.9	Hiperplano con un margen no aceptable.	49
2.10	Hiperplano con un margen aceptable.	50
2.11	Hiperplanos en 2D y 3D [14].	51
2.12	Hiperplano en 3D para clasificación de datos céntricos.	51
3.1	Proceso general para el análisis de sentimientos en Twitter.	53
3.2	Diagrama de casos de uso.	55
3.3	Diagrama de actividades.	56
3.4	Diagrama UML clases de la arquitectura del sistema	57
3.5	Página para registro como desarrollador	59
3.6	Diagrama de clases para el pre procesamiento de documentos	80
3.7	Salida en terminal de Python al completar el preprocesamiento y entrenamiento.	93
4.1	Resultado de las pruebas realizadas.	97

Lista de tablas

2.1	Estructura de una matriz de confusión.	17
3.1	Promedio del desempeño de SVM	92
3.2	Matriz de confusión obtenida del SVM.	93
4.1	Promedio del desempeño de SVM	98

Lista de códigos

2.1	Ejemplo de formato XML.	21
2.2	Ejemplo de formato JSON.	22
2.3	Uso de lower() y upper() con cadenas en Python	25
2.4	Separación de una cadena en elementos con split()	25
2.5	Implementación del método replace()	25
2.6	Algunos métodos del tipo de dato str	25
2.7	Símbolos del módulo string de Python	26
2.8	Creación de listas en Python	27
2.9	Agregando elementos a una lista con append()	27
2.10	Vaciando una lista en Python con el método clear()	28
2.11	Ejemplo de uso del método copy() de las listas en Python	28
2.12	Conteo de elementos en una lista de Python	28
2.13	Agregando elementos a una lista a otra lista con el método extend()	28
2.14	Obtener el índice de un elemento	29
2.15	Insertando elemento en posición específica en una lista	29
2.16	Extracción de elemento en una lista con el método pop()	29
2.17	Eliminando elemento de una lista	29
2.18	Invirtiendo orden de una lista en Python	30
2.19	Ordenando una lista con el método sort()	30
2.20	Sentencia for usando range() en Python	31
2.21	Sentencia for sobre una lista de Python	32
2.22	Iteración sobre una lista con while	32
2.23	Obteniendo el mayor de dos números con sentencia if	34
2.24	Iteración definida por estructura condicional if else	34
2.25	Evalando una variable con elif	35
2.26	Identificación de binarios pares e impares en Python.	36
2.27	Ejemplo del manejo de la excepción AttributeError	39
2.28	Ejemplo del manejo de la excepción ZeroDivisionError y else.	39
3.1	Clase Credenciales	61
3.2	Bibliotecas y Configuración para autenticación con Tweepy	61
3.3	Contenedores para recolección de datos	62
3.4	Descarga de tweets con Tweepy	63
3.5	Obtención de texto de los tweets y retweets	64

3.6	Remueve emojis y marcas en el texto	65
3.7	Limpieza básica con método TweetCleaner	66
3.8	Filtro de duplicidad	66
3.9	Creación del archivo csv	66
3.10	Creación de clase PreprocesamientoGeneral	69
3.11	Asignación de nombres de archivo diccionario y stopwords.	69
3.12	Funcion para la búsqueda de palabras en un diccionario	69
3.13	Funcion para buscar expresion regular.	70
3.14	Función para sustituir expresion regular.	70
3.15	Creación del diccionario.	70
3.16	Elimina los saltos de linea del archivo y guarda las stopwords.	71
3.17	Verificacion palabras en la lista de stopwords.	71
3.18	Clase para remover stopwords	72
3.19	Remueve stopwords desde el diccionario de stopwords	73
3.20	Remueve los signos de puntuación de una cadena	73
3.21	Remueve los acentos en una cadena.	74
3.22	Remueve los nombres de usuarios.	75
3.23	Remueve los URL en una cadena	75
3.24	Clase de procesamiento previo para archivos de texto.	76
3.25	Elimina stopwords desde a clase preprocesamiento.	77
3.26	Elimina acentos desde a clase preprocesamiento.	77
3.27	Cambia las URL desde a clase preprocesamiento.	78
3.28	Cambia nombres de usuario desde a clase preprocesamiento.	78
3.29	Cambia signos de puntutacion desde a clase preprocesamiento.	78
3.30	Carga ruta del archivo de datos.	78
3.31	Lectura del archivo de datos.	79
3.32	Bibliotecas para métodos de la clase ModelFuntions.	82
3.33	Constructor de la clase ModelFunctions	82
3.34	Método cleanText e ingreso de archivo de preprocesamiento.	83
3.35	Separación de contenido de set de datos.	84
3.36	Homologación de nombres de usuario en tweets.	84
3.37	Remoción de signos de puntuación y stopwords del texto de los tweets.	85
3.38	Reintegración de contenedores de datos	86
3.39	Remoción de tweets neutrales	86
3.40	Contenedores de stopwords de Spacy y signos de puntuación	88
3.41	Definición del metodo myTokenizerCleaner	88
3.42	Construcción de pipeline del modelo.	89
3.43	Definición de set de datos para entrenamiento y test	90
3.44	Entrenamiento y calculo de métricas del SVM	91
3.45	Implementación de precisión mínima de la SVM	91
3.46	Impresión de métricas del SVM	92
3.47	Modelo con entrenamiento terminado.	92
4.1	Importación de módulos para realizar pruebas.	95

4.2	Método cleanText e ingreso de archivo de preprocesamiento.	95
4.3	Carga y lectura de conjunto de pruebas.	95
4.4	Ejecución e impresión de pruebas del modelo basado en SVM.	96

Capítulo 1

INTRODUCCIÓN

Han pasado más de tres años del inicio de la pandemia por COVID19. Sin lugar a dudas, la mayoría de las personas no estábamos preparadas para afrontar esta situación. Aunque en la historia de la humanidad, ya han habido pandemias por otras enfermedades, como Peste Negra, Viruela y Gripe Española, entre otras, una de las mayores diferencias de la última pandemia por COVID19 con respecto de las anteriores, fue la existencia de las tecnologías de la información y comunicación (TICs).

Mediante las TICs, logramos estar comunicados con familiares y amigos, sin importar las ubicaciones geográficas de cada uno. Además, las TICs nos permitieron estar informados sobre riesgos, recomendaciones, y otros aspectos importantes relacionados a la enfermedad. Las redes sociales, parte importante de las TICs, favorecieron el poder expresar nuestro sentir y pensar en esos momentos tan complicados.

Los mensajes publicados por las personas durante la pandemia por COVID19, a través de las redes sociales, forman ya parte de la historia digital. Estas publicaciones pueden ser analizadas mediante técnicas de análisis de sentimientos, para mostrar emociones que expresamos a través de textos en redes sociales, en ese episodio de la línea del tiempo de la humanidad.

1.1 Planteamiento del problema

La pandemia por COVID19 afectó de diversas maneras al mundo entero, entre ellas, en lo económico, político, sanitario y en los estados emocionales de millones de personas. Estos estados se reflejaron de varias formas, por ejemplo, en las opiniones o comentarios que usuarios de las redes sociales publicaron.

En esta tesis, se aborda el problema de identificar cuáles fueron los tipos de emociones que manifestaron los usuarios de la red social Twitter, enfocándose en los mensajes en idioma Español. Para resolver el problema se aplica el análisis de sentimientos.

1.2 Objetivos

1.2.1 Objetivo general

Aplicar análisis de sentimientos a publicaciones relacionadas con el término COVID19 recolectadas en Twitter, para identificar las principales emociones.

1.2.2 Objetivos específicos

Los objetivos específicos de esta tesis son los siguientes:

1. Presentar la terminología y técnicas generales para análisis de sentimientos.
2. Identificar las principales características y funcionalidades de la biblioteca Spacy para análisis de sentimientos con Python.
3. Recolectar publicaciones relacionadas con el tema de COVID19.
4. Aplicar análisis de sentimientos a las publicaciones recolectadas.

1.3 Hipótesis

Se parte de la hipótesis que la mayoría de las publicaciones sobre el tema de COVID19 en español, contienen sentimientos de miedo o angustia, mientras que hay una minoría de publicaciones en las que se puede identificar sentimientos de alegría y felicidad. Para probar esta hipótesis, se aplicará el proceso de análisis de sentimientos, identificando también lugares, personas y organizaciones más frecuentemente mencionadas.

1.4 Justificación

El análisis de sentimientos es una herramienta poderosa para identificar polaridades sobre algo, alguien o alguna situación. El reciente problema de salud debido a COVID19, ha tenido una fuerte repercusión en todo el mundo, y en todas las áreas. Parte del sentir de la población con respecto a este problema es capturado en las publicaciones en redes sociales, o en servicios de *microbloging*, siendo Twitter la plataforma más representativa de este tipo.

El analizar los sentimientos en publicaciones en español sobre el tema de COVID19, además de ser un tema actual y estudiado por investigadores en todo el mundo, permite entender nuestro entorno local y global, utilizando para ello tecnologías y conceptos vistos durante la formación profesional como ingeniero en computación, lo que representa una valiosa oportunidad de aplicar los conocimientos.

1.5 Alcance

El análisis de sentimientos se limitará a las siguientes restricciones:

- Las publicaciones serán en idioma español solamente.
- Se realizará un pre-procesamiento a los datos con código propio, para limpiar los textos principalmente.

- Se aprovecharán las capacidades de la biblioteca Spacy para realizar la mayor parte del análisis de sentimientos, sin embargo, también se implementarán algunas partes con código propio.
- Se probará la hipótesis principal de la tesis, para lo cual se calcularán frecuencias de los sentimientos encontrados en las publicaciones.

Capítulo 2

MARCO TEÓRICO

En este capítulo se describen las tecnologías involucradas para el análisis de sentimientos en idioma español. El lenguaje de programación utilizado es Python, por lo que se describen las características principales y fundamentos usando en este trabajo.

2.1 COVID19

El COVID19, también conocido como coronavirus, es una enfermedad infecciosa causada por el virus SARS-CoV-2. Se transmite principalmente de persona a persona a través de gotículas respiratorias cuando alguien infectado tose, estornuda o habla. Los síntomas más comunes incluyen fiebre, tos seca, fatiga, dificultad para respirar, dolor de garganta y pérdida del gusto o el olfato. Sin embargo, es importante destacar que los síntomas pueden variar desde leves hasta graves, y algunas personas pueden ser asintomáticas.

La pandemia de COVID19 ha tenido un impacto significativo en diversas áreas, incluyendo la economía, la educación y las finanzas. En términos económicos, un incontable número de empresas se han visto obligadas a cerrar o reducir sus operaciones debido a las restricciones impuestas para contener la propagación del virus. Esto ha llevado a una disminución en la producción, pérdida de empleos y caída de los ingresos en la mayoría de los países del mundo. Los sectores más afectados han

sido el turismo, la hostelería y el comercio minorista.

En cuanto a la educación, la pandemia ha generado el cierre prolongado de escuelas y universidades en todo el planeta. Esto ha obligado a la implementación de la educación a distancia, con clases virtuales y aprendizaje en línea. Sin embargo, esta transición ha presentado desafíos significativos, especialmente en regiones con acceso limitado a la tecnología y recursos educativos. Además, la falta de interacción social y el cierre de actividades extracurriculares han afectado el bienestar emocional y social de los estudiantes.

En el ámbito financiero, la pandemia ha provocado una volatilidad significativa en los mercados financieros. Las bolsas de valores experimentaron caídas abruptas durante los primeros meses de la crisis, generando incertidumbre en los inversionistas y afectando los fondos de pensiones y las carteras de inversión. Los gobiernos y los bancos centrales han implementado medidas de estímulo económico para mitigar el impacto negativo en la economía, como programas de ayuda financiera y recortes de tasas de interés.

En resumen, el COVID19 es una enfermedad infecciosa que ha tenido un impacto global significativo en diversos aspectos de la sociedad. Además de representar una amenaza para la salud pública, ha causado estragos en la economía, la educación y las finanzas. A medida que avanzamos en la lucha contra esta pandemia, es fundamental seguir las pautas de salud y seguridad recomendadas por las autoridades sanitarias para protegernos a nosotros mismos y a los demás.

El COVID19 ha sido objeto de numerosas conversaciones y expresiones emocionales en las redes sociales. Desde el comienzo de la pandemia, las plataformas digitales se han convertido en un espacio para que las personas compartan sus sentimientos y preocupaciones sobre este tema tan relevante.

En las redes sociales, se han expresado una amplia gama de emociones relacionadas con el COVID19. Uno de los sentimientos más comunes es el miedo. Las personas han compartido su temor ante la incertidumbre de la situación, el riesgo de contagio y la posibilidad de que sus seres queridos se vean afectados. El miedo también se ha manifestado en relación con las restricciones impuestas y el impacto en la vida cotidiana.

Además del miedo, también ha habido una expresión generalizada de frustración y enojo. Las personas han expresado su frustración por las medidas de confinamiento, el distanciamiento social y la interrupción de actividades y eventos importantes. También han mostrado su enojo hacia aquellos que no cumplen con las pautas de salud y seguridad, lo que podría poner en riesgo a otros.

Por otro lado, ha habido un sentido de solidaridad y empatía en las redes sociales. Las personas han compartido mensajes de apoyo y aliento para aquellos que están luchando contra la enfermedad o han perdido a seres queridos. También se han organizado iniciativas para ayudar a las comunidades más vulnerables, como la entrega de alimentos o la donación de suministros médicos.

Asimismo, las redes sociales han sido una plataforma para compartir historias de superación y resiliencia. Personas que han superado el virus han compartido sus experiencias y brindado esperanza a otros. También ha habido muestras de agradecimiento hacia los trabajadores de la salud y otros profesionales esenciales que han estado en la primera línea de batalla.

En resumen, las redes sociales han sido un espacio donde se han expresado una variedad de sentimientos relacionados con el COVID19. Desde el miedo y la frustración hasta la solidaridad y la esperanza, las plataformas digitales han permitido que las personas compartan sus emociones, busquen apoyo mutuo y encuentren formas de hacer frente a esta situación sin precedentes.

2.2 Análisis de sentimientos

Los sentimientos son muy complejos y pueden determinar el comportamiento de una persona. Según la RAE (Real Academia Española), el término sentimiento se define como el "*Hecho o efecto de sentir o sentirse / Estado afectivo del animo*" [30]. Con el simple hecho de hablar con una persona, saludarla e incluso observar sus expresiones, escuchar sus opiniones o comentarios se puede conocer una parte de sus sentimientos, que pueden proveer de información relevante sobre una persona.

Una característica intrínseca de los seres humanos es la dificultad de ocultar sus emociones. Las emociones reflejan el estado de ánimo de una persona, y al analizarlas, se puede obtener información valiosa sobre la misma.

Por otro lado, el término "análisis" tiene diversas definiciones. Según la RAE, se refiere a "*La distinción y separación de las partes de algo para conocer su composición*", al "*Estudio detallado de algo, especialmente de una obra o de un escrito*", y al "*Tratamiento psicoanalítico*" [30].

Aquí tienes una mejora para el texto:

Si nos ceñimos a la primera definición, podemos apreciar que cuando algo puede ser descompuesto en sus partes fundamentales, entonces es susceptible de ser analizado. Por ejemplo, algo tan cotidiano como el lenguaje puede ser desglosado en componentes como sujeto, sustantivo, verbo, entre otros. A partir de esta descomposición, podemos determinar si una persona expresa algún tipo de sentimiento en su lenguaje natural, ya sea en sus comentarios u opiniones.

Con el advenimiento y la popularización del análisis automático del lenguaje hace algunos años, se ha observado que uno de sus objetivos primordiales es extraer información subjetiva del lenguaje natural. Esto se logra mediante herramientas definidas por el análisis de sentimientos [26].

El análisis de sentimientos apareció a principios de la década del año 2000, cuando todavía las redes sociales no figuraban en la comunicación, incluso no existían. Pero el lenguaje natural lleva mas tiempo como una línea de investigación; sin embargo, la invención de las redes sociales potenció su beneficio en cuanto a investigación.

En este trabajo se trata un tema de interés común para personas, empresas y organizaciones: conocer la opinión de la gente. En la plataforma Twitter, uno de los temas más frecuentes a partir de 2020 fue "**COVID19**". Este generó una gran cantidad de opiniones. Cuando se trabaja análisis de sentimientos en redes sociales, se pueden clasificar las publicaciones en tres niveles en cuanto a opiniones o comentarios sobre un producto o sujeto [26]: El nivel de mensaje donde se generaliza el sentimiento como Negativo, Neutral o Positivo en un solo texto; el nivel de oración

donde el sentimiento se determina por cada oración o enunciado en un texto; y el nivel de entidad, que detalla el sentimiento entre un sujeto y un contexto o situación. A continuación se muestran algunos ejemplos.

- Nivel Mensaje: "Un día me subí a un taxi horrible, el asiento estaba muy sucio y el auto era muy feo pero el chofer era muy buena persona, y no me cobró nada por derramar una bebida. A veces usar aplicaciones para transportarnos es más cómodo, pero más susceptibles a generar cargos por incidentes con los usuarios." Tomando como producto un taxi el sentimiento en general sería negativo.
- Nivel Oración: "Un día me subí a un taxi horrible, el asiento estaba muy sucio y el auto era muy feo pero el chofer era muy buena persona, y no me cobró nada por derramar una bebida.". Tomando en cuenta que la persona se queja del producto que en este caso es el taxi el sentimiento sería negativo, en cambio si el producto es el servicio del taxista, el sentimiento sería positivo.
- Nivel Entidad: "Un día me subí a un taxi horrible (negativo), el asiento estaba muy sucio (negativo) y el auto era muy feo (negativo), pero el chofer era muy buena persona (positivo) y no me cobro nada por derramar una bebida (positivo)". Como comentamos anteriormente, este nivel es el que extrae más detalles sentimentales de un texto ya que evalúa cada oración tomando en cuenta un producto.

2.2.1 Métodos para análisis de sentimientos

Un análisis psicológico de los comentarios de una persona con métodos específicos de esa disciplina es más profundo, y puede capturar más detalles acerca de la personalidad de una persona. En este trabajo, el análisis se concentra en los textos de una opinión o comentarios, usando técnicas de aprendizaje automático.

Según la RAE, aprendizaje es "*Acción y efecto de aprender algún arte, oficio u otra cosa y Adquisición por la práctica de una conducta duradera* " [30]. Esto se

menciona porque el objetivo de este trabajo es enseñarle a una computadora a reconocer sentimientos, y para esto el aprendizaje juega un papel muy importante, aunque algunos piensan que es una virtud única de un ser vivo. Sin embargo, para que una computadora pueda aprender a resolver un problema, se necesita integrar a ella la capacidad de procesar información y generar una predicción. Este tipo de aprendizaje es posible gracias a la inteligencia artificial (IA), el aprendizaje automático (machine learning), y como caso particular las redes neuronales (neural networks) y aprendizaje profundo (deep learning).

La gran característica entre ellas es que así como aprendizaje automático es un sub campo de la inteligencia artificial, el aprendizaje profundo es un subcampo de aprendizaje automático pero los algoritmos de aprendizaje profundo casi siempre están compuestos por varias capas de redes neuronales [20]. Además de esto, el aprendizaje automático implica que una computadora puede aprender a realizar un proceso o tarea basándose en datos, sin estar programada específicamente. Como ya comentamos, el aprendizaje profundo se compone mayormente de redes neuronales, y esto no es casualidad ya que esta si este enfocado a simular el pensamiento humano en algoritmos utilizando una estructura muy compleja con datos no estructurados, por ejemplo una simple imagen [40].

El análisis de sentimientos se centra en la identificación de emociones, típicamente en textos, aunque también puede aplicarse a vídeos, voz, imágenes, entre otros medios. Para este propósito, el análisis de sentimientos se vale de una herramienta sumamente útil denominada minería de textos (*Text Mining*). La mayoría de los métodos para realizar análisis de sentimientos incorporan esta herramienta mediante el procesamiento de lenguaje natural, por sus siglas en ingles NLP (Natural Language Processing), dado que proporciona una excelente capacidad para manejar textos mediante diversas herramientas que extraen patrones de un texto.

A menudo se confunde la minería de datos con la minería de textos, pero la diferencia radica en el tipo de análisis que se lleva a cabo. Mientras que la minería de datos se basa en la fuente de datos en general, la minería de textos se enfoca en una colección

específica de documentos.

Si solo extraemos patrones de un texto mediante minería de textos, el usuario podría fácilmente percibir el sentimiento presente. No obstante, el verdadero objetivo de llevar a cabo un análisis de sentimientos mediante programación es permitir que una computadora pueda reconocer estos patrones emocionales. Para ello, el aprendizaje automático ofrece métodos de aprendizaje tales como **supervisado**, **semi-supervisado** y **no supervisado**, los cuales son eficaces en diferentes contextos y escenarios.

El aprendizaje no supervisado se fundamenta en un entrenamiento sin etiquetas que definan una clase específica, sino en conjuntos de datos sin categorización previa. Entonces, ¿cómo se lleva a cabo este proceso? Aunque pueda parecer contradictorio entrenar sin conocer las clases, el aprendizaje no supervisado utiliza conjuntos de datos para buscar similitudes o patrones entre ellos, con el fin de extraer una estructura subyacente que defina uno o varios grupos. De esta manera, si un dato se asemeja a otro, puede detectarse una estructura que permita diferenciarlo del resto.

Este tipo de aprendizaje puede aplicarse a diversas formas de datos, como imágenes, archivos o información numérica. Los algoritmos de agrupamiento, como el K-means clustering, ejemplifican esta técnica. Estos algoritmos, al recibir un número predeterminado de grupos, encuentran similitudes entre los datos según criterios como la distancia entre puntos, lo que permite definir conglomerados distintos [16].

Entre los métodos más destacados de aprendizaje no supervisado se incluyen las redes neuronales no supervisadas, el agrupamiento (tanto K-means como jerárquico), las reglas de asociación (Eclat y A priori) y la reducción de dimensionalidad (núcleo de PCA y distribución de t PCA) [33].

Por otro lado, el aprendizaje semi supervisado opera con un conjunto de datos que incluye tanto una pequeña parte de datos etiquetados manualmente como otra parte sin etiquetar. Durante el proceso de entrenamiento, el sistema etiqueta los datos no etiquetados para complementar la información disponible, utilizando algoritmos de maximización de expectativas. Además, se incorporan detalles que facilitan el

aprendizaje desde el inicio. Este enfoque emplea un conjunto de reglas para asistir en el etiquetado de algunos datos, así como la similitud léxica y el intercambio de palabras para mejorar el proceso de etiquetado [22].

El análisis supervisado es un tipo de aprendizaje que se basa en datos etiquetados con una clase específica. Utilizando estos datos, es relativamente sencillo crear un modelo para desarrollar un clasificador. Sin embargo, a diferencia del aprendizaje semi supervisado del que hablamos anteriormente, en este caso, es necesario etiquetar todo el conjunto de datos, asegurándose de que solo contenga pares de entrada-salida. Esto implica la creación de una función que relacione directamente los resultados con las entradas proporcionadas.

Es importante destacar que, dado un conjunto de datos limitado, puede ocurrir que una determinada salida se obtenga con mayor frecuencia, lo que puede influir en la precisión del modelo. Además, la ausencia de ciertos elementos en el conjunto de entrenamiento puede dificultar la obtención de resultados precisos para entradas no vistas previamente [16].

Existen varios métodos de aprendizaje supervisado, entre ellos se encuentran las redes neuronales supervisadas, el algoritmo KNN (k-nearest neighbors), las máquinas de soporte vectorial, la regresión logística, los árboles de decisión y los bosques aleatorios.

2.2.2 Proceso general para realizar análisis de sentimientos

Podemos visualizar este proceso como una serie de pasos fundamentales, los cuales pueden asemejarse al famoso método científico debido a la importancia de manejar la información recopilada de manera óptima para obtener resultados integrales. Como hemos observado anteriormente, los métodos pueden variar considerablemente según el propósito de la investigación o el experimento, pero todos comparten un proceso básico que todo análisis de lenguaje natural debe seguir. Cada paso implica la implementación de técnicas necesarias para extraer información del autor de cualquier texto, lo que

constituye la esencia misma de la minería de textos.

Los siguientes pasos conforman una estructura que, en algún momento, todo análisis debe incorporar, al menos incluyendo los pasos de extracción de datos y entrenamiento. Si bien cada programador puede definir su propio proceso, un ejemplo notable es el caso de SentiWordNet en 2006, donde Esuli y Sebastiani propusieron un enfoque extendido sobre el análisis de sentimientos con cuatro pasos clave [1].

- Selection Of Seed Set (Selección del conjunto de semillas)
- Creation Of Training Set (Creación del conjunto de entrenamiento)
- Creation Of Classifiers (Creación de clasificadores)
- Synset Scoring (Sincronización puntajes)

Pero en este caso el enfoque es el análisis de sentimientos en Twitter por ello la estructura contempla los siguientes pasos que serían los más comunes en un modelo así.

Recolección de datos

Esta fase puede considerarse como un paso facultativo, ya que no todos los procedimientos de análisis exigen la recopilación de conjuntos de datos. No obstante, es imprescindible en algún momento de la labor analítica que los datos pertinentes sean obtenidos. La recolección de datos, especialmente en el contexto de las plataformas de redes sociales, puede plantear ciertas dificultades debido a las inquietudes relacionadas con la confidencialidad y la privacidad, que suelen ser motivo de preocupación para las corporaciones responsables de estas plataformas. Incluso la adquisición manual de datos con el propósito de entrenar un modelo puede resultar laboriosa y exigente.

Por consiguiente, algunas plataformas ofrecen la posibilidad de otorgar permisos o credenciales para acceder a su información. Sin embargo, este proceso conlleva la necesidad de que la plataforma esté debidamente informada sobre el uso que se dará a la información, lo que a veces puede dificultar el acceso a la misma. Por ejemplo,

una vez que se han solicitado y obtenido las credenciales pertinentes de la plataforma, como sucede en el caso de Twitter, la documentación de la interfaz de programación de aplicaciones (API, por sus siglas en inglés) especifica que los desarrolladores tienen acceso programático a la plataforma. Este acceso otorga a un programador la capacidad de utilizar estas credenciales desde un script para iniciar sesión y acceder a la API de Twitter, lo que le habilita para hacer uso de herramientas exclusivas y avanzadas destinadas a la manipulación de tweets, mensajes, perfiles de usuarios, imágenes y diversas funcionalidades adicionales [37].

Filtrado de datos

Cuando se lleva a cabo la recolección de datos desde aplicaciones o plataformas digitales, es común obtener conjuntos de datos que contienen miles o incluso millones de registros, abarcando una amplia gama de temas e idiomas. Esta diversidad en el contenido de los conjuntos de datos puede requerir la implementación de filtros para su refinamiento. Dichos filtros pueden aplicarse en función de diversos aspectos de los datos, tales como el idioma, el tema, la fecha, la región e incluso los usuarios asociados.

En ocasiones, resulta obligatorio realizar un proceso de filtrado de estos conjuntos de datos con el fin de seleccionar el material óptimo para su uso en investigaciones o análisis. Continuando con el ejemplo de la API de Twitter, esta herramienta proporciona al usuario la capacidad de aplicar filtros directamente durante el proceso de descarga de los datos. De esta manera, el filtrado se simplifica significativamente, asegurando al usuario que obtendrá únicamente los datos relevantes para sus necesidades específicas.

Extracción de datos

La etapa de extracción de datos comienza con un preprocesamiento destinado a preparar el texto para su análisis, con el objetivo de facilitar su manipulación y comprensión. Este paso es fundamental en el proceso, ya que aquí el text mining despliega sus capacidades al máximo, extrayendo toda la información relevante del texto o documento. Este proceso puede comprender varias capas, que varían según el objetivo de la investigación. Sin embargo, en el análisis de sentimientos, una de las

técnicas más comúnmente utilizadas es la tokenización, que consiste en identificar y separar las partes del texto mediante el uso de expresiones regulares, permitiendo manipular cada palabra individualmente en el análisis [19].

Otra técnica importante es la eliminación de palabras vacías (stopwords), que son términos que no aportan significado al análisis, como pronombres y artículos gramaticales. Asimismo, el proceso de derivación (stemming) implica obtener derivaciones de palabras que pueden ser más útiles en el análisis que las palabras raíz. Existen dos tipos principales de analizadores morfológicos, conocidos como stemmers: aquellos de morfología flexiva, que establecen relaciones sintácticas entre varias palabras de un texto, y los de morfología derivativa, que generan nuevas derivaciones de la palabra raíz, incluso contrarias gramaticalmente [18].

La lematización, por su parte, se refiere a la identificación de la raíz u origen de un token. A diferencia del stemming, la lematización considera la estructura gramatical del idioma, y puede utilizar uno o más diccionarios para interpretar las palabras. Por ejemplo, considerando las palabras universo, universal y universidad, el resultado del stemming sería "univers", mientras que la lematización puede identificar formas verbales o sustantivos válidos en el idioma, apoyándose en diccionarios [2].

Estas técnicas representan algunas de las capas que se aplican en el proceso de extracción de datos. No siempre se utilizan todas ellas, ya que esto dependerá de la necesidad de obtener información relevante del texto. Sin embargo, su aplicación puede resultar de gran ayuda en determinados contextos analíticos.

Análisis de contenido

En esta etapa, el análisis de sentimientos cobra una alta relevancia, ya que es el momento de identificar qué tipos de palabras contiene el texto y etiquetarlas adecuadamente. Es necesario determinar los verbos, adjetivos, sustantivos, personas, lugares, productos, entidades, entre otros aspectos relevantes para poder captar opiniones sobre un producto o tema específico. Este proceso se conoce como etiquetado de partes del discurso (Part of Speech Tagging), o PoS por sus siglas en inglés, y se basa en la tokenización con delimitadores de oraciones [18]. El etiquetado

de partes del discurso desempeña un papel crucial en esta etapa, ya que constituye la base del análisis.

Como ya mencionamos anteriormente, existen varios métodos para llevar a cabo el análisis de sentimientos, ya que más allá de simplemente conocer la etiqueta que tiene una palabra en el texto, lo que da vida al proceso es un motor que implemente un clasificador o red neuronal capaz de realizar predicciones. Estos métodos pueden ser supervisados, no supervisados o semi supervisados.

Para ello, se puede implementar un vector de características con la estructura de un sentimiento, para ser usado en por un clasificador, como una maquina de vectores de soporte SVM por sus siglas en ingles (Support Vector Machine) o una red neuronal con capas específicas diseñadas para identificar sentimientos a través de sus pesos sinápticos. También se pueden utilizar clasificadores como Naive Bayes, KNN o algoritmos de agrupamiento como K-means clustering. Es importante destacar que el análisis puede combinar más de uno de estos métodos, y la elección de la forma de analizarlo dependerá del programador y, sobre todo, de los objetivos de la investigación en cuestión.

Pruebas y revisión

En ocasiones, los datos recopilados no están etiquetados con un sentimiento de referencia respecto a su contenido. Por lo tanto, al realizar un aprendizaje semi o completamente supervisado, es necesario asignar etiquetas manualmente por parte de una persona física o del propio programador. También es posible que el etiquetado sea realizado por el algoritmo, lo que constituye aprendizaje no supervisado. Sin embargo, existe la posibilidad que el programador cometa errores al asignar las etiquetas correctas a los datos, lo que puede llevar a palabras con sentimientos contrarios a ser agrupadas en una categoría equivocada. Esto a su vez puede desencadenar predicciones incorrectas por parte del modelo. Por lo tanto, es crucial revisar el conjunto de entrenamiento (data training) y corregir cualquier error cambiando las etiquetas por las correctas.

Las pruebas sobre un modelo de Análisis de Sentimientos (AS) son una tarea fundamental, pero de gran importancia, ya que ningún sistema de machine learning es 100% efectivo. Para evaluar la eficacia de un modelo predictivo o clasificador, se

pueden utilizar métricas de evaluación como la matriz de confusión, el recall, la precisión, la exactitud, el F1-Score, entre otras [6]. Estas métricas tienen como objetivo principal proporcionar una percepción clara sobre cómo está funcionando el sistema en términos de clasificación y predicción. Utilizando esta información, es posible mejorar el modelo realizando pruebas hasta lograr los porcentajes de error y precisión óptimos. Para lograr esta mejora, se pueden agregar más datos de entrenamiento, mejorar el etiquetado de partes del discurso (PoS) u optimizar la remoción de las palabras vacías (stopwords).

La dinámica de estas métricas es relativamente simple. Por ejemplo, en el caso de la matriz de confusión, esta se caracteriza por su simplicidad, ya que normalmente se aplica en predicciones binarias o de múltiples clases. En la matriz, las filas representan las predicciones realizadas por el modelo, mientras que las columnas muestran las etiquetas correctas. La matriz de confusión se basa en cuatro términos: FP (Falso-Positivo), FN (Falso-Negativo), TN (Verdadero-Negativo) y TP (Verdadero-Positivo), los cuales indican si la predicción fue correcta o incorrecta en cada caso [21]. Estos términos son fundamentales para comprender la calidad de las predicciones del modelo y guiar los ajustes necesarios para mejorar su desempeño. La estructura general de una matriz de confusión se muestra en la tabla 2.1.

		Condición Real	
		+	-
Predicción	+	[TP] Verdadero Positivo	[FP] Falso Positivo
	-	[FN] Falso Negativo	[TN] Verdadero Negativo

Tabla 2.1: Estructura de una matriz de confusión.

El Recall, recuperación o sensibilidad se trata de un cálculo que representa el porcentaje de positividad de una prueba con una condición dada. En otras palabras el este termino indica que tan probable es que una clasificación sea asignada por el hecho de tener ciertas características de una clase [21]. También se considera al Recall

como la tasa de aciertos [18]. El cálculo es muy simple, la ecuación 2.1 muestra cómo se realiza.

$$Recall = \frac{TP}{TP + FN} \quad (2.1)$$

La precisión es una métrica de evaluación ampliamente utilizada y considerada efectiva, al igual que el recall, es de cálculo sencillo. Además, la precisión se puede conceptualizar como la relación "señal a ruido" [18]. Esta métrica representa el porcentaje de predicciones positivas que son correctas, es decir, la frecuencia con la que las conjeturas positivas son acertadas. De manera similar, la especificidad indica el porcentaje de predicciones negativas que son correctas [6]. Las ecuaciones 2.2 y 2.3 ilustran la forma de calcular la precisión y la especificidad, respectivamente.

$$Precisión = \frac{TP}{TP + FP} \quad (2.2)$$

$$Especificidad = \frac{TN}{TN + FN} \quad (2.3)$$

El F1-score, también conocido como puntuación F, representa la media armónica entre la precisión y el recall. Esta métrica proporciona una medida de rendimiento en el rango de 0 a 1, donde un valor de 1 indica una relación perfecta entre las puntuaciones de precisión y recall. El F1-score ofrece una visión resumida de la eficiencia del modelo evaluado y es sensible a las variaciones en el rendimiento. Debido a que esta métrica ignora los verdaderos negativos (TN), se la ha denominado proporción de concordancia positiva. La ecuación 2.4 presenta la forma de calcular esta medida escalar.

$$F1-score = (2) \frac{(Precisión)(Recall)}{(Precisión + Recall)} \quad (2.4)$$

2.2.3 Bibliotecas para análisis de sentimientos

Las bibliotecas desempeñan un papel fundamental en el proceso de análisis, ya que facilitan la programación de tareas de NLP, como la identificación de entidades, la gramática, la tokenización, la eliminación de palabras vacías y la provisión de corpus de datos, entre otras funcionalidades. Aunque existen numerosas bibliotecas disponibles para una variedad de lenguajes de programación, nos centraremos en resaltar las más populares en el contexto de Python, que es uno de los lenguajes más utilizados para tareas de NLP.

Dos de las bibliotecas más destacadas en este ámbito son SpaCy y NLTK. Estas bibliotecas, también conocidas como módulos, ofrecen una amplia gama de funcionalidades para el procesamiento de texto y análisis lingüístico. Además, proporcionan una diversidad de corpus de datos que son fundamentales para el entrenamiento y la evaluación de modelos de NLP.

La instalación de estas bibliotecas, así como de cualquier otra biblioteca en Python, se realiza típicamente desde la línea de comandos o desde la consola del Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) utilizado para programar en este lenguaje. Una vez instaladas, estas bibliotecas pueden ser importadas en los scripts de Python y utilizadas para realizar una amplia gama de tareas de NLP con facilidad y eficiencia.

NLTK

El Natural Language Toolkit (NLTK) es una biblioteca de Python para el manejo de lenguaje natural muy usado por sus beneficiosas características como segmentación de palabras, análisis gramático, clasificación, y muchas mas tareas avanzadas. Además integra API's, Part of the Speech, análisis de sentimientos, interfaces y corpus de texto [4]. En fin todo lo que se necesita para crear una aplicación de NLP. Su instalación es muy sencilla desde linea de comandos basta con usar el comando.

```
1 pip install -U nltk
```

Esta biblioteca es una de las mejores para experimentar en NLP pues la variedad de herramientas en las API's y los corpus de texto que cuentan con colecciones de texto etiquetado y estructurado que permiten realizar un pre-procesamiento usando dichas colecciones como diccionarios para remover stopwords, signos, símbolos, etc. Realmente es muy versátil y no por nada es uno de los mas comunes entre los desarrolladores de Python en NLP [4].

SpaCy

SpaCy es un módulo o biblioteca de código abierto en Python que destaca por su grado industrial y su enorme potencial en el campo del Procesamiento del Lenguaje Natural. Esta biblioteca ofrece soporte para más de 60 idiomas y se distingue por su agilidad y eficiencia, gracias a sus modelos pre-entrenados y sus características exclusivas.

Una de las ventajas más destacadas de SpaCy es su capacidad para trabajar con diferentes frameworks, como Tensorflow y la API de Keras. Esta versatilidad permite desarrollar algoritmos de NLP complejos de manera más sencilla y eficiente, incluso en comparación con otros lenguajes de programación.

SpaCy es ampliamente utilizado en aplicaciones de aprendizaje automático del mundo real, como el análisis de sentimientos, el reconocimiento de intenciones y la extracción de contexto. Una de las características más destacadas de SpaCy es su facilidad para crear tuberías (pipelines) personalizadas, donde el texto puede pasar por varios procesos de text mining y se pueden agregar componentes personalizados en la posición deseada.

En resumen, SpaCy es una biblioteca completa que ofrece una amplia gama de herramientas y funcionalidades para la creación de aplicaciones de NLP, lo que la convierte en una opción popular entre los desarrolladores en este campo [2].

Para instalar SpaCy, basta con dirigirse a la página de documentación de SpaCy NLP y navegar hasta la sección de "usage", donde se ofrecen varias opciones de descarga disponibles para Linux, MacOS y Windows. El usuario puede personalizar su descarga de acuerdo a sus preferencias, seleccionando el administrador de paquetes, el hardware,

las entradas, la plataforma y el tipo de tubería (pipeline) que desee utilizar [10].

Las siguientes líneas de código descargan e instalan SpaCy NLP en un entorno Linux. Se configura para trabajar únicamente con la unidad central de procesamiento (CPU), aunque también se ofrece la opción de aprovechar una tarjeta de vídeo, en caso de estar disponible. Se utiliza **conda** como administrador de paquetes, y se descarga el modelo pequeño de tuberías pre-entrenadas en español, con una estructura eficiente.

```
1 conda install -c conda-forge spacy
2 python -m spacy download es_core_news_sm
```

2.3 JSON

El formato **JSON** (*JavaScript Object Notation*) surgió a finales de los 90, cuando el intercambio de datos comenzó a volverse complejo. El lenguaje **XML** (*Extensible Markup Language*), similar a **HTML** (*HyperText Markup Language*), era la mejor opción para los desarrolladores. Con el paso de los años se volvió popular entre desarrolladores un nuevo formato para el intercambio de datos, el cual es más sencillo, ágil y sobre todo más elegante, además de ser inspirado en un objeto de JavaScript: JSON. Este último presenta una facilidad de gestión de datos más con respecto a XML, ya que no utiliza etiquetas o marcas, en cambio de esto se implementa en tuplas de datos. Un ejemplo realizado en XML y otro en JSON se muestran en los Códigos 2.1 y 2.2. Como puede observarse, XML usa etiquetas que se identifican con los paréntesis triangulares, mientras que JSON usa tuplas con pares de datos "campo": "valor" [31].

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Clientes>
3     <Cliente id="1">
4         <Nombre>Esteban Lira Alanis</Nombre>
5         <Cuenta>400262</Cuenta>
6         <Telefono>5560176501</Telefono>
7         <Direccion>Calle Rayon</Direccion>
8     </Cliente>
9 </Clientes>
```

Código 2.1: Ejemplo de formato XML.

```
1 {  
2   "Clientes": [  
3     {  
4       "Nombre": "Esteban Lira Alanis",  
5       "Cuenta": "400262",  
6       "Telefono": "5560176501",  
7       "Direccion": "Calle Rayon"  
8     }  
9   ]  
}
```

Código 2.2: Ejemplo de formato JSON.

XML todavía se sigue usando, pero ante la popularidad de JSON entre los desarrolladores modernos, este último se volvió el formato sucesor de XML y el preferido de intercambio de datos entre el servidor y el navegador. Esto llevó consigo que un gran número de lenguajes de programación implementaran bibliotecas y soporte para este formato. Así, ahora es posible usarlo en casi todos los lenguajes de programación de alto nivel como Python, Java, C++, C#, JavaScript , etc [17].

2.4 Lenguaje de programación Python

La historia de este lenguaje de programación se remonta al año de 1991, cuando un hombre llamado Guido Van Rossum de Holanda, quería diseñar un lenguaje de sintaxis simple y entendible, parecido al lenguaje natural del idioma inglés. Con esta idea desarrolló Van Rossum el lenguaje de programación Python [9].

Entre muchos lenguajes de programación Python es uno de los más comunes en los desarrolladores de hoy día. En marzo de 2022 Python aparece como el lenguaje número uno en la lista publicada por la compañía Tiobe [36]. Python presenta varias ventajas frente a otros lenguajes, convirtiéndolo en uno de los mejores para empezar a programar, y también para seguir programando. Cabe resaltar que las ventajas pueden ser relativas ya que cada lenguaje generalmente se crea para un propósito en especial y su eficiencia esta completamente relacionada con el objetivo de su desarrollo.

Desde que Python 1.0 fue lanzado en 1994 el lenguaje ha pasado por varias versiones hasta nuestros días, posicionándonos en la versión 3.9 lanzada en octubre de 2020. En los últimos 20 años Python ha adquirido mejoras notables en cada nueva versión. Esto no es casualidad, pues Python goza de una calidad incomparable según desarrolladores, en gran parte por su legibilidad y comprensión.

El hecho de ser multiplataforma y de código abierto (*open source*) hace que Python se encuentre alcance de múltiples usuarios de diferentes sistemas operativos en distintos entornos e interfaces. Es un lenguaje de propósito general, que soporta desde el desarrollo de videojuegos, ciencia de datos, desarrollo Web, entre otras. Elimina la mayoría de la sintaxis que puede confundir a un desarrollador novato, por ejemplo, no se usan puntos y comas al final de las sentencias, no hay necesidad de usar paréntesis en las estructuras de control de flujo de programa y no se usan llaves para especificar bloques de código. Python tiene una enorme cantidad de bibliotecas alto nivel para diferentes propósitos [9].

Este lenguaje podría decirse que tomó lo mejor de los lenguajes base en su desarrollo como (*C/C++ y Java*), pues soporta el paradigma estructurado y orientado a objetos. Python puede ejecutar código de C y también usar sus bibliotecas.

De acuerdo con Lutz [23], Según los usuarios de Python este es uno de los lenguajes de programación más fáciles de aprender por varios factores como calidad de software, productividad, portabilidad, soporte, integración de componentes y el hecho de disfrutarlo.

A continuación se presentan algunos elementos importantes de este lenguaje que fueron usados para el desarrollo de esta investigación.

2.4.1 Cadenas en Python

Python, como cualquier otro lenguaje trabaja con múltiples tipos de datos integrados, fundamentales o incorporados. A diferencia de otros lenguajes, en Python las listas, diccionarios, tuplas, conjuntos y cadenas de caracteres son tipos de datos integrados.

Las cadenas son secuencias de Python que tienen métodos para su manipulación, ya que son objetos. Las cadenas de Python son esencialmente objetos conformados por un conjunto de caracteres no mutables formando así una cadena de texto.

Las cadenas (**str**) pueden contener letras, números, símbolos, caracteres especiales, y en las mas recientes versiones de Python se pueden incorporar emojis. Los principales usos se enfocan en contener información como oraciones, incluso se usa implícitamente en la función **print** de Python para imprimir en pantalla. Casi todos los programas y desarrolladores no necesariamente de Python han usado las cadenas de texto en sus códigos [9] [29].

La principal característica de estas cadenas de texto es su manipulación por medio de índices o las posiciones relativas de izquierda a derecha de los elementos en la cadena, el hecho de ser una secuencia significa que esta mantiene un orden cuando se almacenan y se extraen. Los índices hacen referencia a direcciones de memoria donde se aloja la información de dicha cadena. Las cadenas de Python pueden hacer uso de diversos métodos que facilitan su gestión. [23]

La Figura 2.1 muestra la estructura de una cadena en Python. Como se puede apreciar una cadena de texto se comporta básicamente como un arreglo de caracteres.

Ejemplo	[	C	a	d	e	n	a		d	e		t	e	x	t	o	]
Ejemplo	[	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	]

Ejemplo = "Cadena de texto" Ejemplo[9] = " " Ejemplo[12] = "x"

Figura 2.1: Cadenas de texto en Python.

A continuación se describirán algunos de los métodos más comunes en cadenas de texto de Python. Tales como convertir mayúsculas a minúsculas y viceversa con `lower()` y `upper()` respectivamente ejemplo en el Código 2.3.

```

1 cadena = "UAEMEX Zumpango 2022"
2 print(cadena) # imprime: UAEMEX Zumpango 2022
3 cadena = cadena.lower()
4 print(cadena) # imprime: uaemex zumpango 2022
5 cadena = cadena.upper()
6 print(cadena) # imprime: UAEMEX ZUMPANGO 2022

```

Código 2.3: Uso de lower() y upper() con cadenas en Python

Para dividir una cadena en elementos, se usa el método split(), esto es muy útil en el análisis de sentimientos, ya que es necesario procesar cada palabra de un documento de manera separada. Véase el Código 2.4.

```

1 cadena = "Hola mundo UAEM Zumpango"
2 tokens = cadena.split() # tokens es una lista ["Hola", "mundo",
3                           "UAEM", "Zumpango"]
4 print(tokens)

```

Código 2.4: Separación de una cadena en elementos con split()

Con del método replace() podemos substituir algún elemento de una cadena como una letra o número por otro. El Código 2.5 ejemplifica esto.

```

1 cadena = "Hola Mundo"
2 print(cadena) # Imprime : "Hola Mundo"
3 Resultado = cadena.replace("Hola", "H014")

```

Código 2.5: Implementación del método replace()

A veces, es necesario verificar si una cadena representa un dígito, o si solo contiene letras, si es mayúscula o minúscula, etc. El Código 2.6 muestra algunos métodos para realizar este tipo de comprobaciones.

```

1 cadena = "Su direccion es Arboleda #125"
2 print(cadena) #imprime : "Su direcccionees Arboleda #125"
3 # Verificamos si el lemento 5 es una letra,
4 Resultado = cadena[5].isalpha()
5 print(Resultado) #imprime : True
6 #Verificamos si el elemento 28 es un digito
7 Resultado = cadena[28].isdigit()
8 print(Resultado) #imprime : True
9 #Verificamos si el elemento 26 es un simbolo
10 Resultado = cadena[26].isascii()
11 print(Resultado) #imprime : True

```

```

12 #Verificamos si el elemento 16 es una minuscula
13 Resultado = cadena[16].islower()
14 print(Resultado) #imprime : False

```

Código 2.6: Algunos métodos del tipo de dato str

Módulo string

Cuando se trabaja con cadenas de texto, es común tener que identificar algunos elementos como espacios (barra espaciadora, tecla de tabulador, etc.), dígitos, signos de puntuación y letras mayúsculas o minúsculas, entre otros. En lugar que hacer todo desde cero, puede usarse el módulo **string** de Python que ya contiene muchos de los elementos mencionados anteriormente [28]. El Código 2.7 muestra estas constantes.

```

1 import string
2 # imprime abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
3 print(string.ascii_letters)
4 print(string.ascii_lowercase) # imprime
5   abcdefghijklmnopqrstuvwxyz
6 print(string.ascii_uppercase) # imprime
7   ABCDEFGHIJKLMNOPQRSTUVWXYZ
8 print(string.digits)          # imprime 0123456789
9 print(string.hexdigits)       # imprime 0123456789abcdefABCDEF
10 print(string.whitespace)     # imprime ' \t\n\r\x0b\x0c'
11 # imprime !"#$%&'()*+,-./:;?@[\\]^_`{|}~
12 print(string.punctuation)

```

Código 2.7: Símbolos del módulo string de Python

2.4.2 Listas en Python

A diferencia de otros lenguajes como C/C++ y Java, Python no usa arreglos o estructuras de datos que son parte de paquetes especiales. En su lugar, usa listas enlazadas como uno de sus tipos de datos fundamentales. Una lista es una colección de cero o más elementos/objetos que pueden ser del mismo o de diferente tipo, es decir es heterogénea lo que hace esta estructura más versátil a diferencia de los arreglos en otros lenguajes. Además esta secuencia es mutable y hace uso de métodos incorporados como lo hace una cadena de texto [23] [9].

Las listas son usadas ampliamente en los programas en Python, por lo que conviene dar un repaso a la forma de crearlas, acceder a sus elementos y sus principales métodos, para acceder a sus elementos al igual que las cadenas podemos hacer uso de índices positivos como negativos, es decir es que tomen de referencia el inicio o final respectivamente [34].

El Código 2.8 muestra las formas de crear listas en Python. Las líneas 2 y 4 muestran dos maneras de crear listas vacías, es decir, que no contienen elementos inicialmente. En la línea 6 se muestra cómo crear una lista con varios elementos. Observe que estos elementos pueden ser de distinto tipo.

```
1 #Creacion de una lista vacia
2 listaA = []
3 #Creacion de una lista vacia
4 listaC = list()
5 #Creacion de una lista con elementos
6 listaB = ["ICO", 'UAEM', "Zumpango", 2022, 3.14]
```

Código 2.8: Creación de listas en Python

Las listas de Python tienen los siguientes métodos principales: `append()`, `clear()`, `copy()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove()`, `reverse()` y `sort()`. La funcionalidad de cada uno de estos métodos es muy intuitiva. Se mostrarán fragmentos de código para explicar algunos de los métodos anteriores.

Uno de los métodos más usados de las listas en Python es **`append()`**, que agrega elementos al final de la misma, el Código 2.9 contiene un ejemplo de este método.

```
1 # Ejemplo Append
2 Lista = [1,3,4,6]
3 Lista.append(5)
4 print(Lista)           # Imprime [1,3,4,6,5]
```

Código 2.9: Agregando elementos a una lista con `append()`

Para eliminar todo el contenido de una lista se usa el método **`clear()`**. Cuando se imprime una lista vacía se visualiza un par de corchetes, como se muestra en el código 2.10.

```

1 # Ejemplo Clear
2 Lista = ["U", "A", "E", "M"]
3 Lista.clear()
4 print(Lista)          # Imprime: []

```

Código 2.10: Vaciando una lista en Python con el método clear()

El método **copy()** de las listas enlazadas de Python regresa una copia de la lista que lo invocó. Esto es útil si queremos modificar una copia de la lista, pero no la lista original. El Código 2.11 muestra un ejemplo de este método.

```

1 # Ejemplo Copy
2 ListaA = []
3 ListaB = ["ICO", 'UAEM', "Zumpango", 2022, 3.14]
4 ListaA = ListaB.copy()
5 print(ListaA)          # Imprime: ['ICO', 'UAEM', 'Zumpango', 2022,
                        3.14]

```

Código 2.11: Ejemplo de uso del método copy() de las listas en Python

En ocasiones es necesario calcular la frecuencia de aparición de elementos en una lista, para esos casos se puede usar el método **count()**, que devuelve el número de objetos que son de igual valor al indicado en el argumento proporcionado. El Código 2.12 nos deja ver la forma de usar este método.

```

1 miLista = ["a", 1, 2, "a", "b", "c"]
2 print(miLista.count("a")) # imprime: 2

```

Código 2.12: Conteo de elementos en una lista de Python

El Código 2.13 muestra como se usa el método **extend()** para agregar al final de una lista los elementos del iterable proporcionado, que puede ser otra lista.

```

1 # Ejemplo Extend
2 ListaA = ["CU", 23]
3 ListaB = ["ICO", 'UAEM', "Zumpango", 2022, 3.14]
4 ListaA.extend(ListaB)
5 print(ListaA)          # Imprime: ["CU", 23, "ICO", "UAEM",
                        "Zumpango", 2022, 3.14]

```

Código 2.13: Agregando elementos a una lista a otra lista con el método extend()

Para obtener el índice de un elemento en específico basta con usar el método **index()**, como el Código 2.14 lo implementa. Es importante mencionar que si no existe el elemento en la lista, se lanzará una excepción al ejecutarse.

```
1 # Ejemplo Index
2 Lista = ["U","A","E","M"]
3 print(Lista.index("E"))      # Imprime: 2
```

Código 2.14: Obtener el índice de un elemento

Si deseamos colocar un elemento en una lista de Python en una posición en específico, se puede usar el método **insert()**, como se muestra en el Código 2.15, se debe de indicar la posición y el elemento a insertar.

```
1 # Ejemplo Insert
2 Lista = ["U","A","E","M"]
3 Lista.insert(2,"2022")
4 print(Lista)          # Imprime: ['U', 'A', '2022', 'E', 'M']
```

Código 2.15: Insertando elemento en posición específica en una lista

En el Código 2.16 con el método **pop()** se muestra cómo extraer un elemento de la lista especificando el índice.

```
1 # Ejemplo Pop
2 Lista = ["U","A","E","M"]
3 print(Lista.pop(1))      # Imprime: A
4 print(Lista)             # Imprime: ['U', 'E', 'M']
```

Código 2.16: Extracción de elemento en una lista con el método pop()

El método **remove()**, permite eliminar un elemento de la lista si proporcionamos el contenido de dicho elemento como en el Código 2.17.

```
1 # Ejemplo Remove
2 Lista = ["U","A","E","M"]
3 Lista.remove("E")
4 print(Lista)          # Imprime: ['U', 'A', 'M']
```

Código 2.17: Eliminando elemento de una lista

En los Códigos 2.18 y 2.19 se implementan un par de métodos que modifican el orden de los elementos, estos son **reverse()** para invertir el orden de los elementos, y **sort()** que nos permite ordenar los elementos de una lista.

```
1 # Ejemplo Reverse
2 Lista = ["U","A","E","M"]
3 Lista.reverse()
4 print(Lista)      # Imprime: ['M', 'E', 'A', 'U']
```

Código 2.18: Invirtiendo orden de una lista en Python

```
1 #Metodo Sort
2 lista = [2,6,8,4,7,9]
3 lista.sort()
4 print(lista)      # Imprime: [2, 4, 6, 7, 8, 9]
```

Código 2.19: Ordenando una lista con el método sort()

2.4.3 Estructuras de control de flujo en Python

Para lograr que un programa de computadora pueda tomar decisiones, se usan las estructuras de control de flujo. Como su nombre lo indica, estas pueden controlar el flujo de un programa dependiendo de ciertas condiciones basadas en los valores de ciertas variables. Existen tres distintos tipos de sentencias de control de flujo, las cuales son: *Secuenciales*, *Condicionales* y *Repetitivas*. Por estructura de control secuencial, se quiere dar a entender que las instrucciones o sentencias son ejecutadas en el mismo orden en que están escritas en el código fuente y solo deben tener un punto de entrada y un punto de salida [5].

Todos los lenguajes de programación de alto nivel cuentan con estas estructuras de control de flujo, con una sintaxis diferente pero similar entre ellos. A continuación se describen los tipos de sentencias junto con ejemplos y sus diagramas de flujo.

Sentencias de control de flujo de programa de repetición

Un programa puede contener secciones donde es necesario usar bucles o ciclos de repetición. Las sentencias repetitivas pueden ejecutar fragmentos de código el número

de veces que el programador necesite. Un cantidad que se define a priori o hasta cumplir una cierta condición. Las principales sentencias repetitivas de Python son las siguientes [7].

Estructura de control for : Esta sentencia puede hacer repeticiones previamente definidas es decir se conoce cuantas iteraciones se realizaran [24]. La sentencia for en Python tiene una estructura simple y muy intuitiva. Usa dos componentes, una variable y un objeto sobre el cual se pueda iterar. La variable toma cada uno de los valores del iterador, en orden sucesivo, hasta llegar al último. Uno de estos iteradores es **range()** que proporciona progresiones numéricas. Otros iteradores o estructuras que permiten ser recorridas son las listas, diccionarios, Archivos, cadenas y otras más [9].

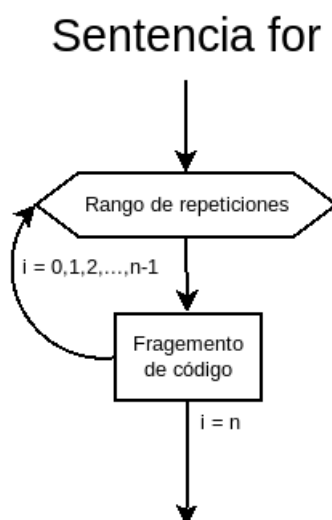


Figura 2.2: Diagrama de flujo de la sentencia for.

En la Figura 2.2 se puede ver el diagrama de flujo, y en los Códigos 2.21 y 2.20 podemos ver un par de ejemplos sobre for.

```
1 # Sentencia for con una progresion nuemrica range() de 1 a 9
2 for contador in range(1,10):
3     print(contador)
4     ""
5 Imprime:
6     1
7     2
8     ...
9     ...
```

```

10     ...
11     8
12     9
13 """

```

Código 2.20: Sentencia for usando range() en Python

```

1 # Sentencia for sobre los items de una lista
2 Lista = ["E1","E2","E3"]
3 for i in Lista:
4     print(i)
5 """
6 Imprime:
7     E1
8     E2
9     E3
10 """

```

Código 2.21: Sentencia for sobre una lista de Python

Sentencia de control while: Este bucle, a diferencia del anterior (for), sólo usa una sentencia condicional y sus iteraciones son indefinidas. En alguna parte del código, casi siempre en el código que se repite, se modifican las variables que componen la sentencia condicional, permitiendo de esta manera que el ciclo termine, de no ser así se tornaría un ciclo infinito [3], esta sentencia tiene una semántica muy simple "Mientras se cumpla cierta condición has esto" [9]. En la Figura 2.3 se puede ver el diagrama de esta sentencia while, y en el Código 2.22 se muestra un ejemplo simple pero completo del uso de la estructura de control while. Es necesario hacer notar que las variables usadas en la sentencia condicional deben de ser inicializadas antes, o se producirá un error en tiempo de ejecución. También es importante observar que en la línea de código 6 modifica la variable i, que hace que el ciclo termine eventualmente. Por último, en algunos programas asíncronos, las variables de la sentencia condicional del while son modificadas en procesos diferentes al actual, como por ejemplo, en hilos. En estos casos no se verá con facilidad que la modificación de las variables en dichas sentencias condicionales.

```

1 # iteracion sobre una lista con while
2 i = 0
3 Lista = [2,8,12,3,6]

```

Sentencia while

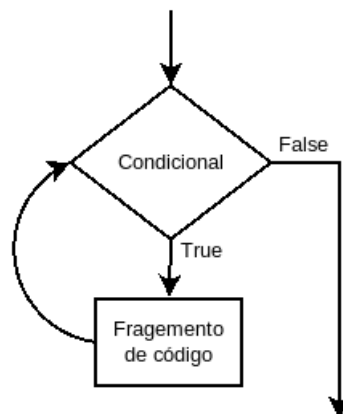


Figura 2.3: Diagrama de flujo de la sentencia while.

```
4 while Lista[i] != 6:
5     print("El elemento ",i," es un ",Lista[i])
6     i = i + 1
7     """
8 Imprime:
9     El elemento 0 es un 2
10    El elemento 1 es un 8
11    El elemento 2 es un 12
12    El elemento 3 es un 3
13    """
```

Código 2.22: Iteración sobre una lista con while

Sentencias de control de flujo de programa condicionales

La base de la programación estructurada es tomar decisiones en el flujo de un programa para alcanzar un objetivo. Este tipo de decisiones implementan sentencias de condicionales o de decisión, pues según se cumplan ciertas condiciones en un código se decide que instrucciones ejecutar y cuales no [5]. En Python las estructuras básicas de este tipo son: **if**, **else** y **elif**. Cada una tiene un uso específico que se describirá a continuación.

sentencia de control if: Esta sentencia puede entenderse en términos naturales como *"si esto es verdadero entonces ejecuta lo siguiente."*, donde "esto" es una sentencia condicional, mientras que "siguiente" es un bloque de código. En otras palabras, se

puede decir que si la expresión que le proporcionamos a la sentencia es verdadera, entonces se ejecutara un bloque de código. Un ejemplo de la sentencia if se puede ver en el Código 2.23.

```
1 # Ejemplo de sentencia if
2 A = 3
3 B = 1
4 if A > B:
5     print("La sentencia es verdadera")
6 # Imprime: La sentencia es verdadera
```

Código 2.23: Obteniendo el mayor de dos números con sentencia if

Sentencia else: La sentencia else trabaja junto con la sentencia anterior (**if**). En caso de que la sentencia condicional sea falsa, puede hacerse que el flujo de programa ejecute un bloque de código específico. Hay que dejar claro que no puede existir una sentencia else sin una sentencia if. El código 2.24 muestra un ejemplo del uso de else con un ciclo implementado con while.

```
1 # iteracion definida por estructura condicional if else
2 i = 0
3 while True:
4     print(i)
5     if i == 20:
6         break
7     else:
8         i = i + 1
9 ""
10 Imprime:
11 0
12 1
13 2
14 ...
15 ...
16 ...
17 19
18 20
19 ""
```

Código 2.24: Iteración definida por estructura condicional if else

elif: Es una sentencia variante de else, que además de depender de un if anterior, puede usarse otra sentencia condicional para tomar otra decisión. Así podemos evaluar varias expresiones una tras otra y ser más objetivos sobre el flujo de un programa. El

Código 2.25 contiene un ejemplo.

```
1 # Ejemplo elif
2 A = 3
3 B = 1
4 if A == 1:
5     print("Variable A = 1")
6 elif B == 1:
7     print("Variable B = 1 ")
8 # Imprime: Variable B = 1
```

Código 2.25: Evaluando una variable con elif

Expresiones regulares

Cuando en un formulario ya sea web, de una aplicación de escritorio o de dispositivo móvil un usuario captura datos con errores, es necesario identificarlos y solicitar al usuario que los corrija. Estos errores pueden genera un problema en el flujo de la información en el código.

Algunos errores son más fáciles de detectar que otros, por ejemplo, para datos numéricos como la edad en años o la talla se puede usar una estructura de control if con una condicional simple. Sin embargo, otros errores son un poco más complicados de identificar, sobre todo en los casos en los cuales los datos deben de tener cierta estructura, como la dirección de correo electrónico, dirección de internet, CURP, etc.

Para este tipo de problemas se puede implementar un metalenguaje para lenguaje regular, llamado expresiones regulares (ER), que para ser todavía más técnicos son la implementación de autómatas finitos descritos de forma algebraica [8]. Este tipo de autómatas pueden usar un motor basado en el patrón como un **NFA** (Autómata Finito No Determinista) o basado en la entrada como un **DFA** (Autómata Finito Determinista) cada uno tienen sus ventajas y desventajas [35], pero su propósito es básicamente el mismo, y pueden ser interpretados por casi cualquier lenguaje de programación de alto nivel, de hecho, un compilador necesita de ER para hacer un análisis sintáctico.

Por otra parte, las ER normalmente son muy útiles para los programadores por el

hecho de poder diseñar patrones para aceptar o filtrar cadenas como URL's, números telefónicos, correos electrónicos, comandos, CURPS, fechas, y otros más.

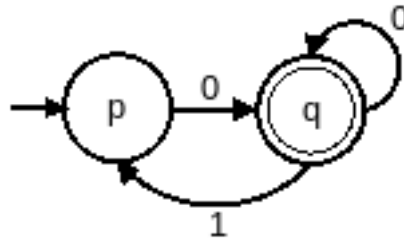


Figura 2.4: Autómata simple para identificar números binarios pares.

Para crear una ER es necesario tener un poco de conocimiento sobre teoría de conjuntos, diagramas de Moore y las operaciones de ER. A continuación presentaremos un ejemplo simple para identificar números binarios y saber si son pares o impares. La Figura 2.4 ilustra un autómata que gráficamente sería la ER a implementar. El estado inicial es q, y el estado final es p. Es bien sabido que los número binarios pares terminan con cero, por lo que se llega al estado final cada vez que se introduce un cero, si se introduce un valor uno después de haber introducido un cero, entonces el autómata pasa al estado p, indicando que el número no es par [35].

En el lenguaje de programación Python, se cuenta con varios motores de ER como **regex**, **re** y **PCRE** para trabajar con expresiones regulares. El ejemplo del Código 2.26 muestra la forma de emplear el módulo **re** para detectar si una cadena contiene solamente ceros y unos, o sea, si es una cadena de bits, y también para identificar si una cadena binaria introducida por el usuario es par impar. La expresión ***** indica que se acepta cualquier cantidad de veces el carácter que la precede. Si se requiere que sea uno o más caracteres, entonces se colocan colocar dentro de corchetes. Entonces, la ER "[0,1]*0" significa que se aceptan únicamente las cadenas que inician con los caracteres 0 o 1, y que al final debe de haber un carácter 0.

```
1 import re
2
3 while True:
4     patron = re.compile(r"[0,1]*0")
```

```

5 test = input("Ingresa un numero binario: ")
6 if re.fullmatch(patron, test):
7     patron2 = re.compile(r"[0,1]*0")
8     if re.fullmatch(patron2, test):
9         print(f"{test} es un binario par")
10        break
11    else:
12        print(f"{test} es un binario impar")
13        break
14 else:
15     print(f"{test} No es un binario intenta de nuevo")
16
17 """
18 Imprime:
19
20 Ingresa un numero binario: 0101012
21 0101012 No es un binario intenta de nuevo
22
23 Ingresa un numero binario: 1010100
24 1010100 Es un numero binario par

```

Código 2.26: Identificación de binarios pares e impares en Python.

2.4.4 Excepciones

Los errores no siempre son malos, metafóricamente *"nos enseñan donde se debe corregir para evitar tropezar con ellos otra vez"*. Los errores en la ejecución de Python generan bloques de código, que se pueden imprimir. Estos se generan cuando se intenta hacer algo que el interprete no entiende o no puede hacer. Esto es coloquialmente la definición de una **excepción**.

Las excepciones son objetos que Python lanza cuando detecta un error, y le comunican al programador o al usuario que existe una situación anómala o un error en la ejecución de las instrucciones. En ocasiones, sin saberlo, las usamos implícitamente, por ejemplo, en un ciclo, una lista o imprimiendo mensajes. Un programa puede manejar excepciones y seguir en ejecutándose pese a errores, si un programa no maneja sus excepciones termina lanzando un mensaje con el rastreo del error [25].

La forma de manejar excepciones en Python es simple. No importa mucho el origen del error, el cual puede ser muy diverso, como desborde de iteradores, archivos inexistentes, variables desconocidas, operaciones matemáticas, etc. [9]. Manejar excepciones es una tarea sencilla, mediante las sentencias **try-except**. De manera

similar que la sentencia **if-else**, no puede haber un **except** sin un **try** precediéndolo.

Como su traducción del inglés lo indica, (try) "*intentar*" se encarga de ejecutar el bloque de código potencialmente puede generar un error o situación extraña (excepción). Inmediatamente después del bloque de código, colocamos la sentencia **except**, a la cual le proporcionaremos el nombre de la excepción que podríamos recibir.

Una sentencia **if-else** es análoga a **try-except**, solo que en lugar de gestionar decisiones esta última trabaja con errores. Se puede agregar la sentencia **else** a **try-except**. Para poder entender esto de una mejor manera, se muestra un ejemplo genérico del manejo de una excepción en el Código 2.27. Por otra parte, el Código 2.28 nos muestra como podemos agregar un **else** a una sentencia **try-except**.

A continuación se presenta una lista de las excepciones estándar más comunes en Python, así como sus causas.

- **Exception** : Es la clase base de toda excepción en Python.
- **TypeError** : Se genera cuando aplicamos una operación inapropiada a un objeto de tipo no compatible.
- **ZeroDivisionError** : Es lanzada cuando tenemos un cero como divisor generando una indeterminación.
- **FileNotFoundError** : Es generada cuando se solicita un archivo o directorio inexistente.
- **ImportError** : Se genera cuando Python encuentra complicaciones o no puede cargar algún módulo.
- **EOFError** : La función **input()** lanza esta excepción cuando llega al final de un archivo y no puede leer más datos.
- **IndexError** : Esta se genera a partir de un subíndice de secuencia que se encuentra fuera de rango.

- **ValueError** : Es generada cuando se recibe en un método u operación un argumento de tipo correcto pero el valor de este no es apropiado.
- **AttributeError** : Se genera cuando se referencia de forma incorrecta algún atributo o existe un error en la asignación.
- **SyntaxError** : Generada cuando el analizador encuentra un error de sintaxis.

Todas las excepciones en Python son una instancia de **BaseException**. Python permite al desarrollador hacer sus propias excepciones creando una nueva subclase de **Exception** o una subclase de esta, como **FileNotFoundError**. No se recomienda hacer nuevas instancias de la clase **BaseException**. [27]

```
1 A = 35
2 try:
3     B = A.append(9)
4 except AttributeError:
5     print("Error de asignacion.")
6 """
7 Imprime:
8
9 Error de asignacion.
10 """
```

Código 2.27: Ejemplo del manejo de la excepción **AttributeError**

```
1 A = 2
2 B = 0.001
3 try:
4     C = A/B
5 except ZeroDivisionError:
6     print("Estas intentando dividir entre 0.")
7 else:
8     print("Se completo la operacion.")
9 """
10 imprime:
11
12 Se completo la operacion.
13 """
```

Código 2.28: Ejemplo del manejo de la excepción **ZeroDivisionError** y else.

2.5 UML

El UML (Unified Modeling Language) o lenguaje de modelado unificado nació a partir del cambio de paradigmas en programación, en década de 1990, durante los primeros años del desarrollo orientado a objetos (DOO). Sus autores, Grady Booch, Ivar Jacobson y Jim Rumbaugh, ya tenían la noción que los métodos de modelado que ellos habían creado (Booch, OMT y OOSE) no eran consistentes uno respecto a los otros, y si no existía un estándar que pusiera sobre la mesa todas las características que ofrecían los ya existentes continuaría un mal entendimiento de los métodos de modelado. Por ello, tal como lo cuentan Ferré Grau Xavier y Sánchez Segura María Isabel, UML llegó a poner fin ala guerra de métodos. En 1996, una gran cantidad de empresas de software importantes y sobre todo los creadores de los modelos Booch, OMT y OOSE crearon UML, tomando lo mejor de estos métodos para dotar al lenguaje recién nacido la capacidad de volverse un estándar para el futuro del modelado [12].

Un lenguaje de modelado esta compuesto por varias características que ayudan a describir un sistema de forma que se posible comprender que se esta tratando de conseguir. Puede contener pseudo-código, código útil, diagramas, tablas, imágenes, descripciones entre otras cosas mas. Estas ultimas son básicamente lo que un lenguaje de modelado ofrece una notación [15].

El tiempo para adaptarte a un lenguaje orientado a objetos como Java es algo abrumador para diversas personas, sobre todo si se aprendió a programar con lenguajes estructurados como C. El cambio de paradigma es algo tedioso, pero necesario para aprender a usar las ventajas que un lenguaje orientado a objetos presenta, como citan Fowler, Scott y Tom Hadfield: *"Los lenguaje orientados a objetos permiten varias ventajas pero no las proporcionan"* [13].

UML es un lenguaje que permite un modelado de procesos de desarrollo que ayudan en múltiples aspectos a la forma de comprender y plasmar la orientación a objetos. Alguien ajeno al desarrollo creería que el simple hecho de hacer los diagramas o dibujos más bonitos no aportaría nada al desarrollo, pero al contrario, este con base

a la cantidad de modelos y notación que implementa facilita la comprensión cuando se plantea el objetivo y se escribe código de DOO.

Dada la gran cantidad de modelos que el UML ofrece, nos limitaremos a comentar algunos de los modelos que se utilizan en este trabajo.

- Diagrama de clases.
- Diagrama de casos de uso.
- Diagrama de actividades.

2.5.1 Diagrama de clases

Estos diagramas son el mejor modelo que se puede usar para visualizar el DOO, puede describir todo tipo de sistemas orientados a objetos, y las diversas clases con su respectiva relación entre las clases del sistema. Este diagrama es muy intuitivo por su estética y la facilidad de comprender las relaciones que existen entre objetos y entre clases de objetos [13].

Este diagrama provee de una notación muy simple, donde se puede visualizar todos nombres de clase, atributos (variables), métodos (funciones), tipos de dato y en ocasiones modificadores de acceso (publico "+", privado "-", protegido "#", estático "_" y abstracto "*"), entre otros. A continuación se comentaran brevemente las relaciones más comunes en los diagramas de clase en la Figura 2.5 podemos ver como se representan en el modelo.

Un diagrama de clases comprende **asociaciones** que se refieren a la relación entre instancias de clases, para que se pueda ver la dirección de la relación junto con su multiplicidad, la cual indica la cantidad de objetos que participan en una relación (1..1, 1..∞, ∞..1 y ∞..∞) [13].

La agregación y composición son de gran polémica, en especial agregación, esto porque en ocasiones puede confundirse con la asociación. Para esto, UML contiene a la composición que es la variante que nos ayuda a definir que un objeto es parte de un

todo, y si este no existe puede generar que el todo falle [13]. Por ejemplo, imaginemos una computadora donde la memoria RAM es un componente que forma parte de una computadora funcional, y si ella no esta la computadora no encenderá, por otro lado la asociación nos indicaría si una computadora tiene uno o varios usuarios y si no tiene no habría problemas pues seguiría funcionando.

La **dependencia** básicamente es el hecho que una clase dependa de otra directamente. Por esto, si una clase cambia puede que la clase dependiente igual pueda haber sufrido cambios [12].

La **generalización** es muy parecida a la **clasificación**, con la diferencia que la generalización es transitiva [13]. Podemos decir que "Un arquitecto es una persona" y "Una persona es un ser vivo". Así podemos decir que todo arquitecto es una persona pero no todo ser vivo es un arquitecto, por eso los dos enunciados de ejemplo son una generalización y una clasificación respectivamente.

La **implementación o realización** se puede comprender como la acción de tomar lo que necesita una clase o de una interfaz, así puede usar las partes que sean convenientes. Imaginemos una clase zapato, este puede implementar ciertas interfaces según el uso del zapato. De esta manera un zapato de trabajo, un zapato deportivo o un zapato de baile generalizan o particularizan la noción de un zapato.

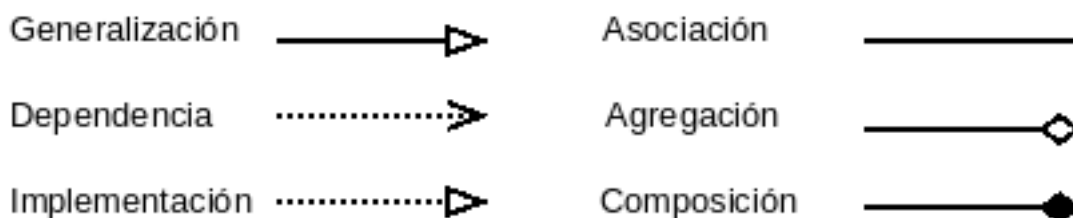


Figura 2.5: Relaciones UML en diagrama de clases.

Los diagramas de clases se caracterizan por gozar de amplia información en sus modelos, pues comprenden además de lo que ya abordamos, atributos, relaciones, polimorfismo, etc. Por ello pueden llegar a volverse muy complejos. Este trabajo específicamente las Figuras 3.4 y 3.6 de las Secciones 3.1.1 y 3.4.1 respectivamente, se puede ver como dos diagramas de clases describen el mismo sistema. Se diferencian

por ser uno muy resumido para facilitar la comprensión y otro con todo lo que un diagrama de clases debe contener para complementar dicha comprensión del sistema.

2.5.2 Diagrama de casos de uso

Este tipo de diagrama UML usa agentes o usuarios externos llamados **actores**. Este modelo proporciona una vista completa del comportamiento de un sistema en cuanto a procesos y gestión de datos. Su notación es más sencilla que la del diagrama de clases, dado que no se enfoca demasiado en cómo están constituidas las clases. Esto no quiere decir que sea menos útil, pues cada diagrama tiene un propósito, y el del **diagrama de casos de uso** es el de tratar de plasmar las funcionalidades del sistema. Cada una de estas funcionalidades son consideradas como un caso de uso, y se usa para visualizar el comportamiento del sistema sin prestar tanta atención al contenido de las clases.

Los **casos de uso** son transacciones o interacciones entre uno o más actores y el sistema. En cada caso se puede ver el flujo de información en el sistema. En un sistema los casos de uso pueden ser dependientes tanto como independientes entre ellos, debido a que esos pueden implementarse entre sí y tratar con objetos compartidos [32].

En el diagrama de casos de uso se dibuja un caso de uso como una elipse. Por su parte un **actor** es todo aquello que puede interactuar con el sistema: personas externas, procesos, entidades, servicios, entre otras. En ejecución, un actor puede estar ligado o relacionado a varios objetos, así como un objeto puede tomar más de un rol en diferentes momentos, esto quiere decir que pueden haber múltiples instancias de actores [32]. Además, participan en varios casos de uso en ejecución intercambiando información entre casos de uso o clases. Los actores en el modelo se dibujan como una persona dibujada con líneas y una cabeza circular, conocido como *stickman*.

Los actores y casos de uso se pueden conectar con relaciones como un diagrama de clases. La relación **asociación** comunica un actor y un caso de uso, y se dibuja con una línea que conecta a estos dos. La relación **incluir y extender** se dibujan como una dependencia de diagrama de clases con la diferencia que se colocan en cada caso las

palabras de esta forma `<<include>>` e `<<extend>>` , uno agrega un comportamiento a un caso de uso desconocido y otro lo agrega a un caso de uso que describe dicha inserción de forma explícita. Por último, la **generalización** es la relación de un caso de uso general y uno en especial que hereda sus características, se dibuja igual que una generalización en un diagrama de clases una línea con una flecha triangular sin relleno que inicia el caso de uso especial y termina el general.



Figura 2.6: Relaciones UML en diagrama de casos de uso.

2.5.3 Diagrama de actividades

Las actividades de un sistema son complejas, sobre todo si estas comparten procesos y clases donde la concurrencia se ve involucrada para llevar a cabo una o más tareas. Los diagramas de actividades permiten ver un sistema desde un punto de vista donde los procesos se encuentran en estados diferentes, según la actividad que realiza. Una **actividad** es aquello que se este modelando, puede ser un proceso tan simple como enviar una carta por correspondencia, hasta casos más complejos como transacciones bancarias.

El inicio de este diagrama se representa con un punto negro, y el final de la mismo se representa con un punto encerrado en un círculo. Los pasos que el diagrama utiliza para modelar se le llaman **acciones y estados de actividad**.

Un ejemplo, como enviar una carta, una acción sería escribir, otra envolver, y una más colocar el destinatario. Estas acciones pueden tener varios propósitos para un sistema orientado a objetos, como clasificar, leer, escribir, creación, control incluso excepciones entre otras.

Los estados de actividad y de acción son cosas que pueden llegar a ser confundidas, por el simple hecho de que se representan de la misma forma en el diagrama (como

un rectángulo redondeado). Sin embargo, la gran diferencia entre ellos radica en el tiempo que relativamente se lleva en su concepción, ya que los estados de acción son instantáneos y no pueden ser interrumpidos; y por otro lado, los estados de actividad pueden descomponerse en más sub-actividades y pueden verse interrumpidos en su proceso [12].

El diagrama de actividades es muy parecido a un diagrama de flujo, de hecho comparte características muy similares como decisiones y fusiones que lo vuelven no secuencial. Las decisiones (o *decision* como se usa en el diagrama) o bifurcaciones son nodos con forma de rombo, donde según el resultado de una condición lógica, se elige seguir un flujo de ejecución. En cada flujo se ejecutan acciones distintas, y eventualmente pueden o no volverse a juntar. Los puntos donde se unen dos o más flujos se llaman fusiones.

Las opciones de la decisión se escriben sobre una línea de transición, entre paréntesis. Como ya comentamos, un estado de actividad puede verse dividido en más sub-actividades o secuencias generadas por decisiones. Las fusiones (merge) unen las actividades que se llevaron a cabo. Puede haber concurrencia en este diagrama, dado que no es secuencial. Las divisiones pueden implementar distintas secuencias de forma concurrente, y unirlos para volver a mantener un sistema secuencial. De esta forma, podemos decir que el asunto se vuelve relativamente secuencial otra vez, para hacer una división se coloca una línea negra en el diagrama e inmediatamente se despliegan las acciones concurrentes. El join o unión se coloca como una línea negra donde culminen las acciones concurrentes. Por otro lado, son llamadas calles o particiones son divisiones en grupos o roles responsables de algunas acciones, estas divisiones hacen que se muestren en columnas.

Esto ha sido un resumen sencillo de un diagrama de actividades, pero es suficiente para comprender como se hace y para que sirve un diagrama de este tipo. En la sección 3.1.1 podemos ver un diagrama de actividades que complementa los diagramas UML utilizados en este trabajo.

2.6 Frecuencia de término - Frecuencia inversa de documento (TF-IDF)

Las siglas **TF-IDF** provienen del término en inglés *Term of Frequency - Inverse Document Frequency*. Este es un concepto muy fácil de entender, ya que se trata de la representación matemática de un documento (páginas web, notas, noticias, opiniones, etc). Esta representación apoya a las computadoras a entender la información, y a usarla para clasificarla, convirtiendo un documento en un vector de características que es mucho más manejable.

Esta técnica esta basada en un concepto anterior de C.J. Cronch, sobre un método de automatizar la creación de diccionarios con palabras que tengan pocas apariciones en los corpus. Ese método usaba un modelo de espacios vectoriales, donde las palabras se convertían en dimensiones. Así, podía hacer proyecciones de los vectores que los documentos proporcionaban [39]. A pesar de que TF-IDF puede trabajar con casi cualquier texto, es conveniente que antes fuesen removidas las palabras vacías (stopwords), signos, y todo lo irrelevante que compone el documento. Dado que trabajar con información sin pre-procesar podría ser poco eficiente.

Esta técnica implementa la bolsa de palabras (Bag of words) la cual es un modelo que utiliza las palabras más comunes en un documento. El cálculo es un proceso muy simple, denominado peso (*weight* en inglés) que consta del producto de 2 partes, la frecuencia de término y la frecuencia inversa de documento.

La ecuación 2.5 muestra cómo se calcula la frecuencia de término que, como su nombre lo sugiere, calcula la frecuencia con la que aparece un término en un documento.

Esta se calcula con la relación de la cantidad de apariciones del término en el documento n_{ik} y la cantidad de términos en el documento d_i [38] [11].

$$TF_{ij} = \frac{n_{ij}}{\sum_0^k n_{ik}} = \frac{n_{ij}}{|d_i|} \quad (2.5)$$

Ahora pasamos a calcular la **IDF**, la cual se trata de una razón logarítmica de una

relación entre el número de documentos de la colección N y el número de documentos en los que aparece dicho término n_j .

Este cálculo es inversamente proporcional a la frecuencia de término, ya que si éste aparece en todos los documentos, entonces tendría un valor 0; y en cuanto a recuperación de información, esto no aporta nada [18]. La ecuación 2.6 muestra como calcular la frecuencia inversa de un documento.

$$IDF_j = \log \frac{N}{n_j} \quad (2.6)$$

Con estas dos ecuaciones, ya sólo falta un detalle por integrar: como comentamos anteriormente en esta sección, el modelo TF-IDF propone un escalar llamado peso, que depende del producto de TF_{ij} e IDF_j . Por esto, la ecuación final que estamos buscando se representa en la ecuación 2.7:

$$TF_{ij} - IDF_j = \left(\frac{n_{ij}}{|d_i|} \right) \left(\log \frac{N}{n_j} \right) \quad (2.7)$$

A continuación se presentará un ejemplo muy simple, pero bastante ilustrativo para dejar más en claro la utilidad de esta técnica.

Recordemos que los documentos pueden ser comentarios, artículos, páginas, etc. Utilizaremos para este ejemplo tres documentos diferentes, pero con un tema en común, en la Figura 2.7 podremos ver una tabla con dichos ejemplos.

DOCUMENTS	
Doc 1	Gatell está muy preocupado porque por "culpa" de los niños que se amparen para recibir vacunas contra el COVID un "adulto" se va a quedar sin la suya. ¿Y las que regalaron a Centroamérica señor? ¿Ahí no se "desprotegieron" mexicanos o cómo?.
Doc 2	A Cuba la critican si vacuna contra la COVID a los niños. Nos dicen que ningún país desarrollado lo está ha hecho. Pero si enviamos a los niños a la escuela sin estar vacunados (como ocurre en Estados Unidos) pues entonces también nos critican. En fin los odiadores y su odio.
Doc 3	La comisión de salud pública aprueba administrar una dosis adicional de la vacuna de coronavirus para algunas personas inmunodeprimidas.

Figura 2.7: Documentos de ejemplo.

Para iniciar, necesitamos calcular TF, la frecuencia con que aparece el término que fue seleccionado del documento. Así lo haremos para cada documento, en la columna TF de la tabla contenida en la Figura 2.8 se muestran estos cálculos en los tres documentos.

Después, procedemos al cálculo de IDF, para este caso la cantidad de documentos son tres, y ahora sólo necesitamos saber en cuántos de estos documentos aparece dicho término. Posteriormente obtenemos la razón logarítmica de la división de los últimos dos conceptos para tener el IDF de cada documento, en la tercer columna de la tabla contenida en la Figura 2.8 se puede apreciar este calculo para cada termino.

El producto de TF e IDF es el escalar que buscamos para cada palabra, en la columna final de la tabla contenida en la Figura 2.8 se encuentran los valores TF-IDF de las palabras de la primer columna en los documentos de la tabla en la Figura 2.7. Si este procedimiento lo realizamos para un cierto número de palabras en un documento, entonces podemos crear un vector que sea la representación de un documento en un espacio vectorial.

Termino	TF (Frecuencia de termino) (Doc 1,Doc 2,Doc 3)			IDF (Frecuencia Inversa de Documento)	TF-IDF (Doc 1,Doc 2,Doc 3)		
vacunas	1/42	0/51	0/20	$\log(3/1)$	0.0262	0	0
COVID	1/42	0/51	1/20	$\log(3/2)$	0.0096	0	0.0202
niños	1/42	1/51	0/20	$\log(3/2)$	0.0096	0.0079	0

Figura 2.8: TF-IDF completo para los 3 documentos.

2.7 Máquinas de Vectores Soporte

Este método de clasificación, también conocido como SVM por sus siglas en ingles (Support Vector Machine), es considerado como uno de los mejores clasificadores binarios. Si bien con TF-IDF ya contamos con vectores muy fáciles de manejar, las SVM nos proporcionan espacios vectoriales para poder diferenciar entre tales vectores.

La clasificación de los datos en SVM lleva a cabo mediante un hiper-plano de margen máximo, en 2 o más dimensiones, donde se puede diferenciar las clases dependiendo en cual semiplano se encuentran.

El hiper-plano que genera este clasificador está compuesto por una cierta cantidad pequeña de vectores, estos son llamados vectores de soporte, y son considerados instancias de entrenamiento más importantes del resto de datos. Este detalle vuelve este algoritmo muy interesante, y uno de los mejores para este tipo de categorizaciones [11].

Por otro lado, el margen más cercano al hiper-plano óptimo es el área que sirve para dividir datos en un espacio vectorial: este margen podemos decir que es una zona muerta de cierta distancia entre cada clase, este requiere ser normalizado para ser útil en la clasificación de la SVM, por su peculiar forma algunas personas le llaman tubo al margen, la distancia o tamaño de este margen es inversamente proporcional a la salida real del valor $x_2 = (x_1, v)$ [16]. Es una parte muy importante en la clasificación de los datos debido a que la distancia entre el hiper-plano y los datos literalmente es la que dicta si una clasificación es más probablemente correcta. Una SVM crea varios prospectos de hiper-planos y evalúa los márgenes de estos hiper-planos para encontrar el apto para realizar la categorización. El hecho de separar los datos en su totalidad no indica que un hiper-plano es el mejor para realizar la tarea como lo podemos ver en la Figura 2.9, ya que puede confundirse muy fácil por su poco margen, de hecho un margen lo suficientemente grande con mínimos errores es más eficaz que uno pequeño sin errores como el de la figura 2.10.

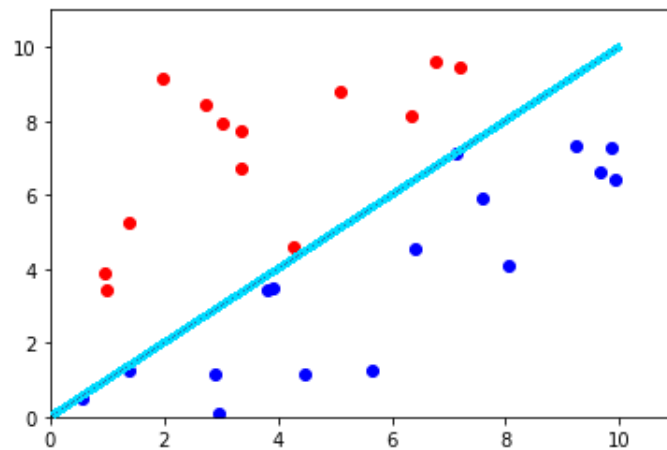


Figura 2.9: Hiperplano con un margen no aceptable.

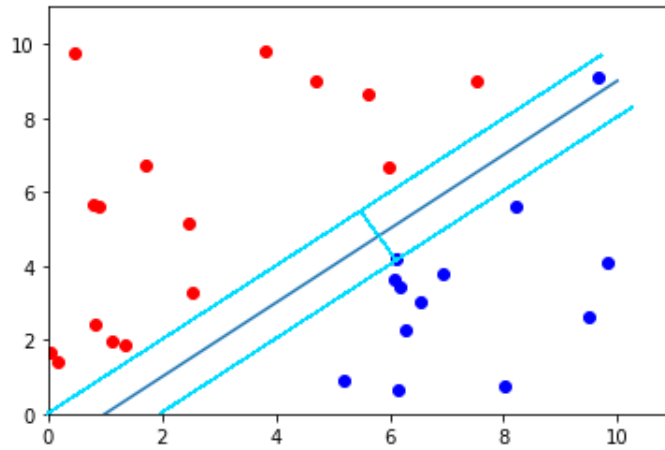


Figura 2.10: Hiperplano con un margen aceptable.

En ocasiones, principalmente en la vida real los datos no son linealmente separables, ¿Qué quiere decir esto?. Significa que no existe un hiperplano lineal que pueda separar los datos de manera perfecta. Los ejemplos mostrados en las Figuras 2.9 y 2.10 son casos linealmente separables.

Ante esto, ¿Qué pasaría si fuese necesario distinguir entre los datos céntricos y los datos mas separados del centro?, Para este tipo de casos existen variantes de SVM, donde se usan diferentes funciones kernel, estas funciones sirven para transformar los datos a diferentes dimensiones para que sea posible hacer una mejor clasificación. Cuando los datos son trasladados a un espacio de dimensiones mas altas un hiperplano se ve diferente.

La Figura 2.11 muestra como se ve un hiperplano en espacios con 2 y 3 dimensiones. La Figura 2.12 por otro lado muestra un ejemplo de un hiperplano de 3 dimensiones, en este se pueden apreciar 2 tipos de clases en el espacio R^3 clasificadas y divididas en $Z = 0.5$ con una función kernel.

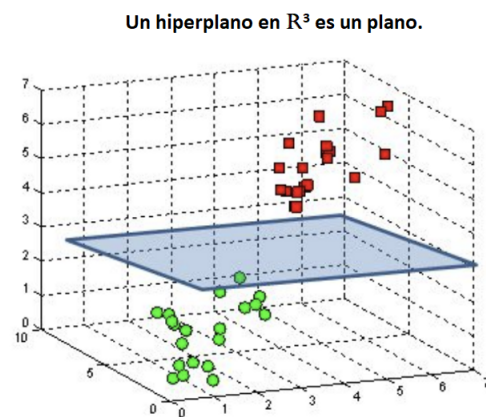
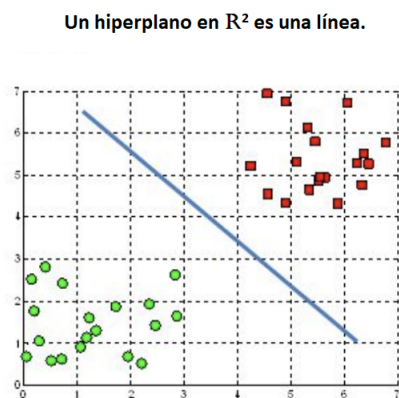


Figura 2.11: Hiperplanos en 2D y 3D [14].

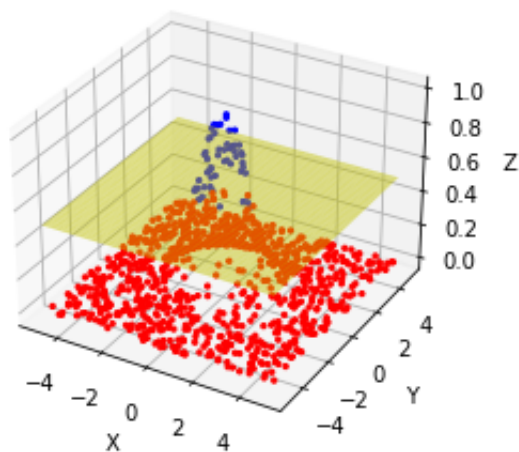


Figura 2.12: Hiperplano en 3D para clasificación de datos céntricos.

Capítulo 3

DESARROLLO

Introducción

En este capítulo se presenta el proceso completo para realizar el análisis de sentimientos sobre publicaciones en Twitter relacionadas con COVID19. Se muestran fragmentos de código (se decidió que los comentarios no tuvieran acentos, por motivos técnicos de presentación con el paquete de $\text{\LaTeX}$ usado para presentar códigos fuente).

3.1 Proceso general para el análisis de sentimientos

En esta sección se presenta de manera general el proceso que se debe llevar a cabo para realizar análisis de sentimientos usando como datos las publicaciones en la red social Twitter. Es necesario indicar en este momento que cada uno de estos pasos serán descritos a detalle en las siguientes secciones. El procedimiento puede resumirse de la siguiente manera:

- El primer paso consiste en solicitar acceso como desarrollador en la plataforma. Para ello se debe de llenar un formulario explicando el motivo por el que se requiere acceso como desarrollador. Twitter responde la solicitud en menos de una semana.



Figura 3.1: Proceso general para el análisis de sentimientos en Twitter.

- Con el acceso concedido, se procede a descargar tweets.
- Los tweets descargados deben de ser sometidos a un proceso de limpieza.
- Se almacenan los datos limpios en algún formato adecuado, en esta tesis se usó el formato de valores separados por comas (csv).
- Se procede al etiquetado manual de datos, identificando tweets positivos, negativos y neutrales bajo un criterio previamente establecido.
- Se discriminan elementos que no contribuyen en la predicción de la polaridad los tweets.
- Se genera un modelo predictivo, extrayendo características de los textos.
- Se evalúa el modelo predictivo.

3.1.1 Diagramas UML de implementación

La Figura 3.1 muestra de manera gráfica el procedimiento explicado anteriormente.

A continuación se presentan los diagramas UML que describen la implementación para el análisis de sentimientos.

En la Figura 3.2 se muestra el diagrama de casos de uso del software a desarrollar y en la Figura 3.3 se muestra el diagrama actividades correspondiente. Estos dos diagramas UML ayudan a entender la dinámica de programación y gestión de datos en este trabajo. Por otro lado también en la Figura 3.4 muestra un diagrama UML de clases simplificado para que sea fácil entender como trabajara el software antes de dar mas detalles puesto que mas adelante se mostrara el diagrama completo.

El código **TweepyDownloader** contiene las **Credenciales** que proporciona Twitter, además de una instancia de **Procesamiento**. Esta última contiene a su vez instancias de clases que le proporcionan funcionalidades como cambiar acentos (**AccentsChanger**), eliminar palabras vacías (**StopWordsRemover**), identificar nombres de usuario y URLs en publicaciones (**UserNamesURL**). Estas últimas clases heredan de **PreProcesamientoGeneral**. La clase **Analisis_Sentimental** contiene métodos para general el modelo predictivo.

En la sección 3.2 se explica en forma detalla la implementación de los principales métodos implementados en las clases. En la sección 3.4.1 se muestra en detalle el diagrama UML de clases.

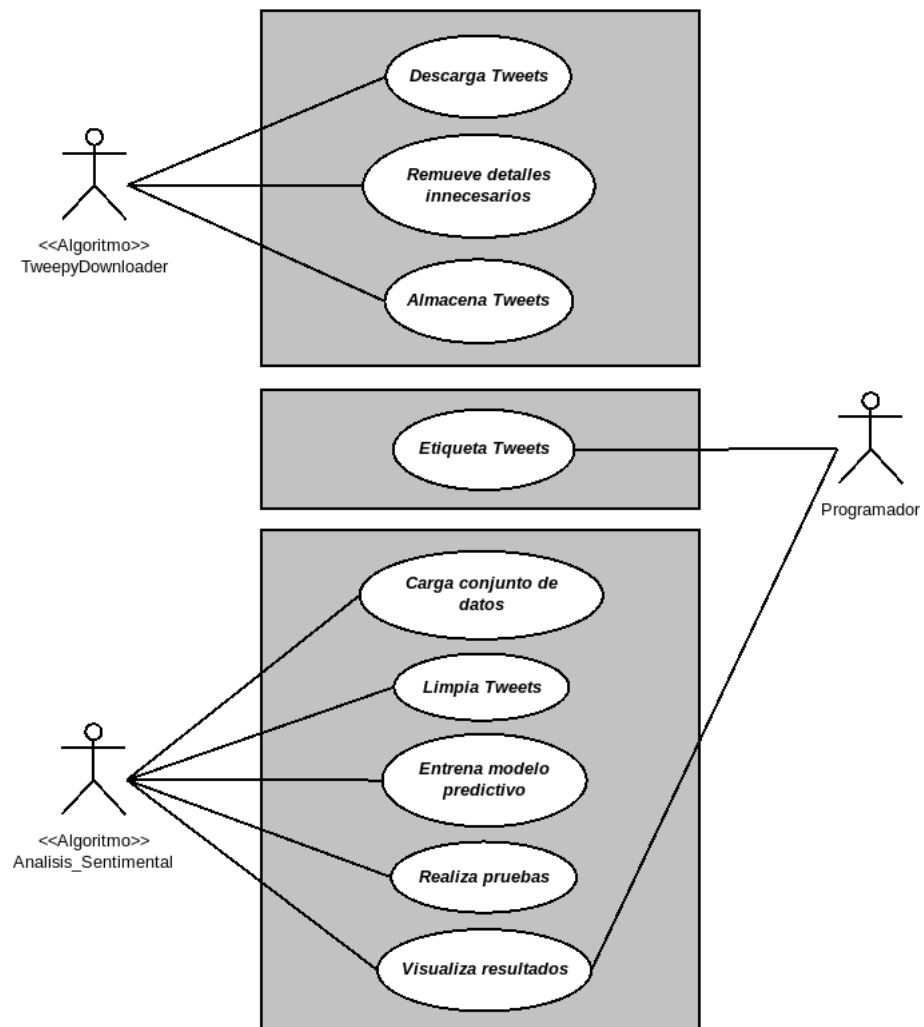


Figura 3.2: Diagrama de casos de uso.

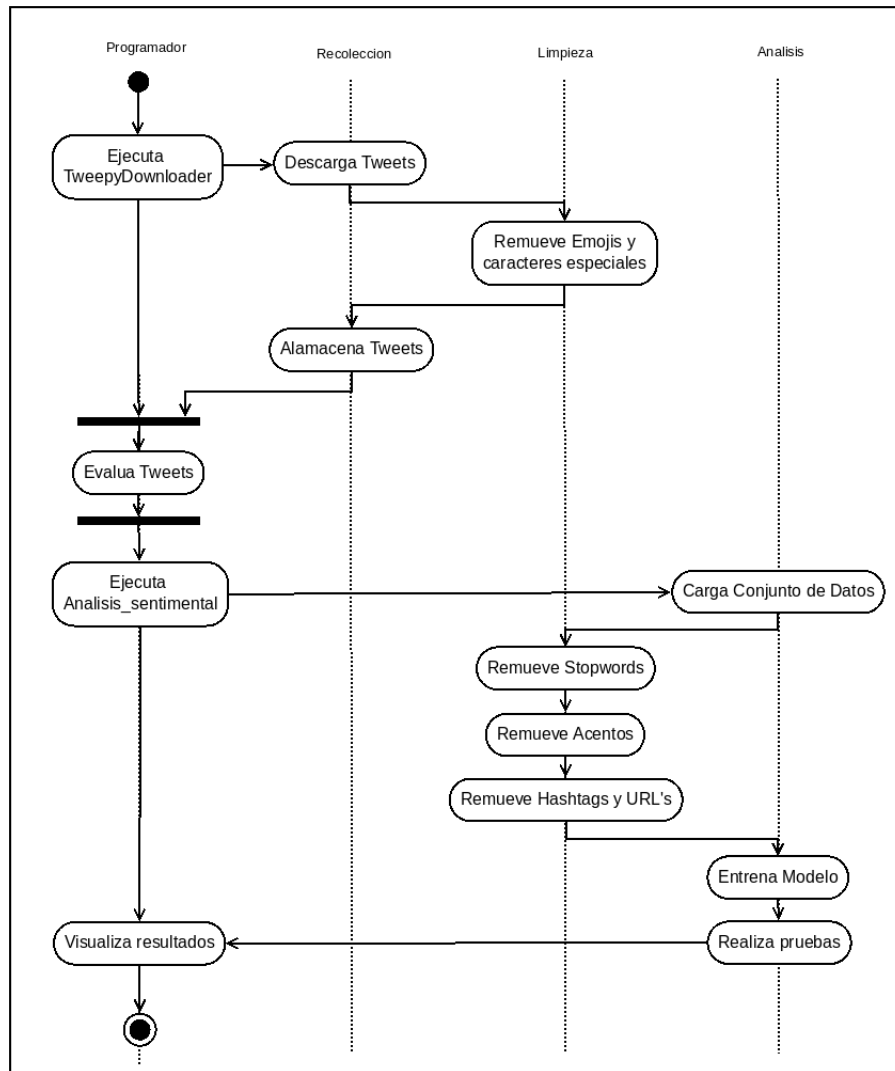


Figura 3.3: Diagrama de actividades.

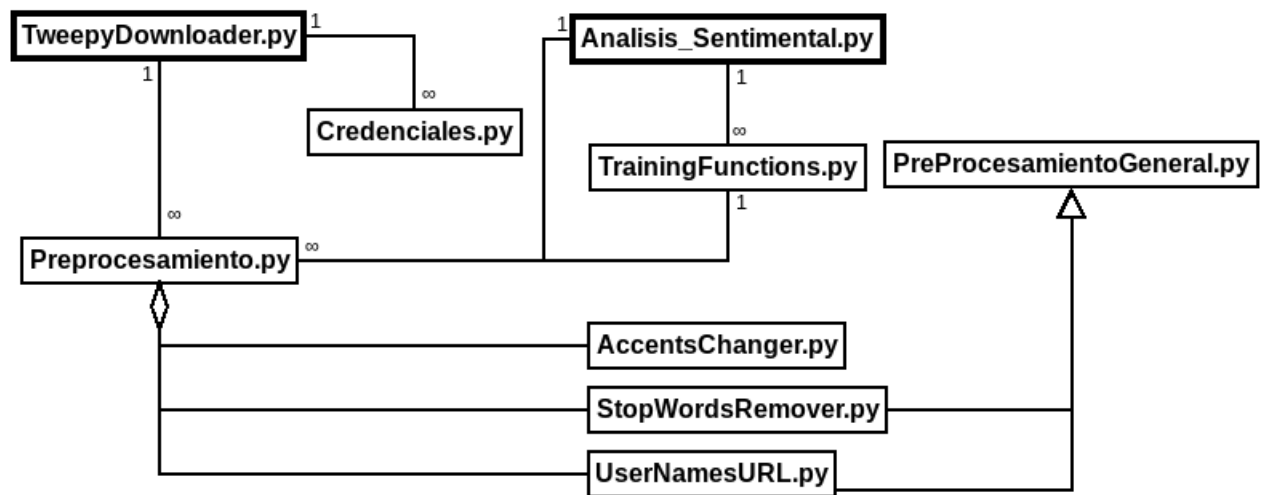


Figura 3.4: Diagrama UML clases de la arquitectura del sistema

3.2 Descripción detallada de los procesos

3.2.1 Recolección de publicaciones en Twitter

Twitter, ahora X, es una red social que cuenta con millones de usuarios en todo el mundo. Diariamente los usuarios de esta plataforma publican sus opiniones sobre hechos o eventos sociales, políticos, atmosféricos, catástrofes, bolsa de valores y de muchos otros temas. Las opiniones publicadas en Twitter reciben el nombre de "tweets".

Los tweets son adecuados para realizar un análisis de sentimientos, ya que en su mayor parte, los usuarios comparte su opinión real sobre algún tema, lo que generalmente conlleva una polaridad, que puede ser positiva, negativa o neutral.

Para realizar un análisis de sentimientos sobre un conjunto de tweets, se puede elegir cualquier tema relevante como punto de análisis. Ejemplos de temas pueden ser hechos históricos, movimientos sociales, incluso una tragedia universal como la segunda guerra mundial, un desastre natural como un terremoto o inundación, o accidentes como el de Chernobyl.

El tema de estudio en este trabajo de tesis será la peor crisis sanitaria de las últimas décadas: la pandemia por COVID19 o Coronavirus. Esta pandemia inició en el año 2019, en Wuhan, China. Al momento que esta tesis se redacta todavía no ha terminado la pandemia. Diariamente se comunican cifras de decesos, vacunación contra este virus, posibles tratamientos, nuevas variantes del virus y medidas sociales por parte de gobiernos o la misma OMS (Organización Mundial de la Salud). El autor considera que por este motivo, es un momento correcto para aplicar un análisis de sentimientos a los tweets sobre COVID19.

Una vez elegido el tema a estudiar, se deben de recolectar tweets relacionados, para lo cual se deben de seguir algunos pasos importantes. Para Twitter la seguridad de los usuarios es prioridad, por lo que para acceder a las APIs (*Application Programming Interface* o en español Interfaz para programación de aplicaciones) de esta red social se requiere de solicitar permisos especiales. Twitter solicita información de la aplicación

a desarrollar, lo más importante es saber quien va usar la información, y qué propósito desea trabajar con dicha información, esto para proteger la confidencialidad de usuarios.

El proceso de solicitud ante Twitter consiste en ingresar a la plataforma de desarrollo, se llena un formulario sobre el uso y la responsabilidad de la información que la organización nos puede dar, normalmente la respuesta en el correo electrónico que se proporciona como contacto tarda al menos un día, dado que se revisa detalladamente cada solicitud. La página de registro es la que se muestra en la Figura 3.5, cuya dirección es <https://developer.twitter.com/en/apply-for-access>



Figura 3.5: Página para registro como desarrollador

Cuando la aplicación es aceptada, Twitter envía un enlace de ingreso a la cuenta de programador, en donde se pueden visualizar las credenciales de acceso a la API de Twitter.

Las credenciales proporcionadas tienen una estructura similar a la muestra a continuación:

- *consumer_Key* = "xxxxxx xxxxxx xxxxxx"
- *consumer_Secret* = "xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx"

- *access_Token* = “xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx
xxxx”
- *access_TokenSecret* = “xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx
xxxxx xxxxx”

Las credenciales recibidas por parte de Twitter permiten acceder a los métodos de recolección de tweets de la API. En esta tesis se usó la biblioteca llamada Tweepy para recolección de los tweets.

Tweepy es el envoltorio de la API de Twitter, es una biblioteca fácil de usar para acceder y recolectar tweets. Tweepy cuenta con gran catalogo de métodos que pueden ayudar la búsqueda y selección de tweets según el programador lo requiera.

3.2.2 Proceso de descarga de tweets

De manera general, el proceso empleado en esta tesis para descargar tweets es el siguiente:

1. Establecer datos de acceso válidos.
2. Autenticarse con los datos de acceso.
3. Preparar contendores para los tweets.
4. Descargar tweets con Tweepy.
5. Filtrar tweets evitando duplicados.

A continuación se presenta la implementación de cada uno de estos pasos.

Datos de acceso

Para comenzar a trabajar con la Tweepy se creó una clase de Python llamada **Credenciales**. Esta clase es usada exclusivamente para contener los datos de acceso. El código 3.1 muestra la implementación de esta clase.

```

1 class Credenciales():
2     consumer_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
3     consumer_secret = '
4     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
5     access_token = '
6     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
7     access_token_secret = '
8     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

```

Código 3.1: Clase Credenciales

Autenticarse con los datos de acceso

En otro script llamado **TweepyDownloader** se creó una instancia de la clase **Credenciales**. Inmediatamente después, se usó el manejador de autenticación de Tweepy, y se le asignan las credenciales de consumidor. Al manejador de autenticación le proporcionamos los token de acceso con el método **setAccessToken()**, para poder usarlo como autor en la conexión con la API de Twitter. La implementación de este proceso de muestra en el código 3.2.

```

1 import Credenciales
2 from Preprocesamiento import Preprocesamiento
3 import tweepy
4 import emoji
5 import numpy as np
6
7 # Credenciales de acceso
8 consumer_key = Credenciales.Credenciales.consumer_key
9 consumer_secret = Credenciales.Credenciales.consumer_secret
10 access_token = Credenciales.Credenciales.access_token
11 access_token_secret = Credenciales.Credenciales.
12     access_token_secret
13
14 # Autenticacion de la API de Twitter
15 auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
16 auth.set_access_token(access_token, access_token_secret)
17 api = tweepy.API(auth, wait_on_rate_limit = True)

```

Código 3.2: Bibliotecas y Configuración para autenticación con Tweepy

Preparar contenedores para los tweets a recolectar

Para almacenar los tweets recolectados se crearon un par de listas, inicialmente vacías. Estas se llaman **TweetsCollection** y **UsefulTweets**. Además, se usó otra lista (llamada **Tags**) para las palabras de referencia usadas en la búsqueda. Los datos que analizaremos serán relacionados con la pandemia de COVID19 que actualmente se vive. También ocupamos un objeto Preprocesamiento referenciado por la variable **p**. El código 3.3 muestra estos elementos.

```
1 # Recoleccion de datos
2 # Etiquetas para la busqueda
3 Tags = ["covid","COVID19","Covid-19","Coronavirus"]
4 UsefulTweets = []
5 TweetsCollection = []
6 p = Preprocesamiento()
```

Código 3.3: Contenedores para recolección de datos

Descargar tweets con Tweepy

Como se mencionó anteriormente, se usó la biblioteca Tweepy para la descarga de documentos (tweets). El método **Cursor()** de Tweepy realiza una búsqueda de tweets, para lo cual es necesario indicarle los datos relacionados a la búsqueda. Los parámetros que se le pasan a este método son los siguientes:

- **api.search**: El método de recolección de la API de Twitter.
- **q** : Las etiquetas para la búsqueda, en nuestro caso se hace referencia a la lista Tag del código 3.3.
- **rpp**: Es la cantidad de tweets que se descargan por página, como máximo para este caso son 100.
- **lang**: Es el lenguaje en el que fueron escritos los tweets. Para nuestro caso lo establecimos a idioma español.

- **tweet_mode**: Indica el modo de descarga para cada tweet, en este caso "extended" evita tweets incompletos.

El objeto **cursor** cuenta con un método llamado **items()** donde se indica la cantidad de tweets a descargar.

Debido que se buscaron tweets relacionados con varios términos (lista Tags en el Código 3.3), es necesario un ciclo para recorrer la lista que los contiene. Se decidió arbitrariamente hacer una recolección de 500 tweets por etiqueta.

```
1 # Ciclo de recoleccion con el metodo search del API
2 while len(UsefulTweets) < 1000:
3     for Tag in Tags:
4         for tweet in tweepy.Cursor(api.search,q=Tag,rpp=100,
5                                     lang="es",
6                                     tweet_mode="extended",
7                                     since="2017-04-03").items
8         (500):
9             TweetsCollection.append(tweet)
```

Código 3.4: Descarga de tweets con Tweepy

NOTA IMPORTANTE

Afortunadamente, los datos usados en esta tesis fueron recolectados antes de que Twitter cambiara de nombre a X. Con las nuevas políticas de X, ahora es necesario realizar un pago para poder usar la API. El procedimiento expuesto anteriormente, todavía es funcional, pero si no se realiza el pago, se recibe el error **Forbidden: 403**.

Extracción de contenido de los tweets

Cuando se ejecuta la descarga de tweets (Código 3.4), lo que se obtiene es una lista de objetos tipo **tweepy.models.Status**. El formato de los datos tipo Status es JSON. Estos datos cuentan con un gran número de atributos de cada tweet descargado, por ejemplo, contiene el nombre de usuario que publicó el tweet, el ID del tweet, la ubicación geográfica, la fecha, el texto, etc.

Para este trabajo, sólo interesa el contenido textual de los tweets, este se encuentra en el atributo **full_text**. En el caso de los objetos que contienen un *retweet* (un tweet

que ha sido replicado por otros usuarios) el campo de texto se verá limitado a 40 caracteres, a esto se le llama un *tweet trunco*.

Recuperar texto completo de tweets y retweets

Para recuperar el texto completo de un tweet se usa el atributo **full_text**. Sin embargo, para los retweets se usa el atributo **retweeted_status.full_text**. Estos atributos son mutuamente excluyentes, por lo que en esta tesis se aprovecha este hecho para recuperación de textos completos de tweets y retweets.

Para proceder a obtener el texto de tweets y retweets se toma en cuenta que los tweets originales a diferencia de los retweets no tienen el objeto status. Si consultamos el atributo **retweeted_status** de un tweet original se lanzará una excepción de atributo, indicando que no existe, por lo que se puede deducir que no es retweet. Con base en esto, si recorremos el conjunto de objetos recolectados, y si se asume que todos fueran retweets, encontraremos el texto integro siempre y cuando si se trate de un retweet. Pero si el objeto es un tweet original, entonces se lanzará una excepción. Podemos entonces implementar una sentencia *try-except* para atrapar el error usar el texto del tweet original. El Código 3.5 muestra cómo se implementó esta estrategia.

```
1  #Obtencion del contenido de tweets
2  for tweet in TweetsCollection:
3      try:
4          TweetText = tweet.retweeted_status.full_text # Es un
Retweet
5      except AttributeError:
6          TweetText = tweet.full_text # Es un Tweet original
```

Código 3.5: Obtención de texto de los tweets y retweets

Limpiar y filtrar tweets duplicados

A pesar de haber recolectado un total de 2,000 tweets, es posible a que puedan haber repetidos. La recolección de datos repetidos en Twitter es un problema común. Esto debido a que si se trata de un tweet escrito por una persona con miles o millones de

seguidores (*followers*), la publicación puede ser retweeteada n cantidad de veces, y por lo tanto es recolectada varias veces. Si bien no podemos evitar que sean recolectados los tweets repetidos, si podemos filtrarlos.

Además del problema de la repetición de tweets recolectados, estos pueden contener caracteres inválidos. Es necesario remover estos, pues no ayudan en el análisis y entorpecen su funcionamiento.

Para proceder a la limpieza de los textos, se hace uso del método **replace** del tipo de dato str incorporado en Python. Este método reemplaza cualquier coincidencia en la cadena del primer parámetro por un segundo str. En este caso lo que se busca es eliminarlos, por lo que basta con usar una cadena vacía. Los elementos a eliminar en el texto son saltos de línea, comillas, o subcadenas como RT, que significa retweet.

Para eliminar los emojis en el texto, en esta tesis se hizo uso de la biblioteca *emoji*. El comando para instalarla es el siguiente:

```
1 pip install emoji
```

Con el uso de la biblioteca emoji se pueden identificar los emojis en las cadenas, y por medio de **get_emoji_regexp** podemos obtener sus expresiones regulares para sustituirlos con **sub** por cadenas de texto vacías. En el Código 3.6 se puede ver la implementación del procesamiento.

```
1 def TweetCleaner(text):  
2     text = text.replace('RT ', '')  
3     text = text.replace('\n', ' ')  
4     text = text.replace('"', '')  
5     text = text.replace("'", '')  
6     text = text.replace(',', '')  
7     return emoji.get_emoji_regexp().sub(r'', text)
```

Código 3.6: Remueve emojis y marcas en el texto

Algunos tweets contienen URLs que son referencia a alguna noticia, imagen, vídeo o anuncio. Estos URLs no son útiles para el motivo de esta tesis, por lo que se deben eliminar. Para ello se implementó un método llamado **changeURL** de la clase **Preprocesamiento**. Véase el Código 3.7. Todas las URLs son sustituidas por la cadena "URL".

```

1 TweetText = TweetCleaner(TweetText)
2 TweetText = "'" + TweetText + "'"
3 TweetText = p.UserNamesURL.changeURL(TweetText, "")

```

Código 3.7: Limpieza básica con método TweetCleaner

Después de aplicar los métodos anteriormente explicados, cada tweet se encuentra limpio, pero todavía no sabemos si este tweet está repetido. Podemos identificar los tweets duplicados si estos los almacenamos en una lista, haremos uso de la lista **Usefultweets** del Código 3.3, aquí se debe implementar un sentencia **if-else** y hacer uso del método **__contains__** pasando cada uno de los tweets, este verificara si el elemento que se le pasa como parámetro ya es parte de la lista en este caso **Usefultweets** y si este no se encuentra en la lista lo agregará, para poder saber cuantos tweets son realmente útiles imprimirá el tamaño de dicha lista. En el Código 3.8 se implementa lo anterior.

```

1 if UsefulTweets.__contains__(TweetText):
2     continue
3 else:
4     UsefulTweets.append(TweetText)
5     print("El texto de ", len(UsefulTweets), "tweets esta
    listo")

```

Código 3.8: Filtro de duplicidad

Archivo csv del conjunto de datos

Los textos de los tweets -también llamados documentos en esta etapa- ya limpios y con lo menos posible de elementos repetidos, pueden ser almacenados en distintos formatos, tales como archivos separados por comas, hoja de cálculo, registros de una tabla en una base de datos relacional, JSON, etc. En esta tesis se hace uso del método **savetxt()** de **numpy** para almacenar todos los documentos en un archivo de valores separados por comas. El Código 3.9 muestra esta implementación.

```

1 # Creacion del archivo de datos

```

```
2 np.savetxt("TweetsData_360.csv", UsefulTweets, delimiter = ",",  
            fmt = '% s')
```

Código 3.9: Creación del archivo csv

3.3 Clasificación o etiquetado del conjunto de datos

Después de obtener la cantidad de tweets suficientes se necesita clasificar todo el conjunto recolectado, esto debido a que en esta tesis se trabajara con un aprendizaje supervisado, esto quiere decir que se trabajara sobre datos previamente etiquetados, cuando estos fueron descargados del API de Twitter no tienen definido un sentimiento en sus atributos de tweet, sino es necesario que algún algoritmo de machine learning (*aprendizaje automático*) lo haga, dado el tema sensible de la pandemia de COVID19 nos parece correcto juzgar cada tweet manualmente y etiquetarlo de la misma forma en el mismo csv.

Para ser objetivos se aplicará una serie de reglas para la evaluación del conjunto de datos, serán etiquetados 500 tweets y estos se clasificarán en 3 sentimientos según sus características enfocadas a lo percibido en el autor de cada tweet, usando *1,0* y *neu* como etiquetas en el archivo. Las reglas se muestran a continuación:

- **Positivo** (*1*) : Se considerará de esta forma a los tweets que muestren alegría o felicidad, también cuando en él se muestre un estado de alegría sin usar sarcasmo, si se notifica un hallazgo científico o acto social que ayude positivamente al estado de la pandemia como vacunas, tratamientos y medidas políticas o preventivas.
- **Negativo** (*0*) : Se considerará así cuando este presente un estado de enojo, sarcasmo, tristeza, también se considerará de esta forma si se plantea noticias malas como cifras de contagios y decesos al alza o cuando simplemente se citan muertes de enfermos COVID19.
- **Neutral** (*neu*) : El tweet será considerado de esta forma si se trata de comentarios donde no se pueda clasificar entre alguno de los anteriores sentimientos.

Ejemplos:

'El último contrato para la adquisición de la vacuna Sputnik V permitirá vacunar a 10 millones de peruanos con las dos dosis requeridas del fármaco.'

Este tweet presenta un sentimiento neutral pues no se percibe enojo o felicidad directamente, pero es una noticia que involucra un acto social como la firma de un contrato para ayudar a un país a combatir la pandemia por medio de vacunas, se podría inferir que el sentimiento del autor es de felicidad por los peruanos.

3.4 Pre-procesamiento de los datos

Para aplicar un análisis de sentimientos, se necesita dar un cierto formato a los tweets clasificados para proceder al análisis. Anteriormente ya se realizó una pequeña limpieza en el Código 3.6, pero esta se enfocaba en detalles del tweet para que pudieran ser almacenados de una forma eficiente. En este punto ya no importan ciertas partes del tweet, pues a pesar de hacer todo lo posible para descargarlos completos, no todo lo que contiene un tweet es relevante para un análisis de sentimientos, existen palabras que no son útiles y solo se usan para dar un sentido coherente a una oración, incluso para **Spacy** resultan inútiles para generar un análisis de lenguaje natural, este tipo de palabras son denominadas *Stopwords*. También es importante deshacerse de detalles puntuales como acentos, hashtags, URL's, nombres de usuario y códigos *UNICODE* que pueden estar en un tweet.

Para comenzar con este preprocesamiento se dividirá en varias clases que a su vez algunas tendrán una clase en común de la cual heredaran métodos y parámetros esta se llamara **PreProcesamientoGeneral**, esto ayudara a ahorrar código en estas clases.

Preprocesamiento General

Esta clase mostrada en el Código 3.10 se ocupará de heredar a algunas clases parámetros y diccionarios necesarios para funciones específicas, así que se crea

principalmente una variable vacía donde mas adelante se encontrara la dirección del archivo diccionario. También se inicializan arrays que funcionaran como tablas para la búsqueda de stopwords.

```
1 import re # Para el regex
2 import numpy as np
3
4 class PreProcesamientoGeneral:
5     #Funcionalidad general de preprocesamiento
6     def __init__(self):
7         # Nombre del diccionario donde se hacen las busquedas
8         self.fileDiccionario = ""
9
10        # Tabla donde se crea el diccionario
11        miDiccionario = np.array([])
12        # Tabla donde se crean las palabras vacias
13        misStopWords = np.array([])
```

Código 3.10: Creación de clase PreprocesamientoGeneral

Después hay que asignar los nombres de archivos que contienen a el diccionario y las stopwords a variables miembro de la clase como se indica en el Código 3.11

```
1     #Funcion para asigna el nombre del archivo de diccionario
2     # a una variable
3     def setDiccionario(self, fileDiccionario):
4         self.fileDiccionario = fileDiccionario
5     # =====
6
7     #Funcion para asignar el nombre del archivo de stopwords a
8     # una variable
9     def setFileStopWords(self, file):
10        self.fileStopWords = file
```

Código 3.11: Asignación de nombres de archivo diccionario y stopwords.

El propósito de crear un diccionario es buscar y verificar la existencia de una palabra en este mismo. El Código 3.12 implementa una función de booleana que indica si una cadena o palabra se encuentra en el diccionario.

```
1     def existsInDicctinary(self, word):
2         for dicWord in self.miDiccionario:
3             if word == dicWord:
4                 return True
5         return False
```

Código 3.12: Funcion para la búsqueda de palabras en un diccionario

Ahora se implementará una función para encontrar coincidencias en cadenas de texto y poder sustituirlas, aquí nos resulta útil la biblioteca **re** por su facilidad de uso en métodos de expresiones regulares. Se usará una serie de métodos de esta biblioteca para manipular y examinar estas expresiones, **replaceRegexWith** será el nombre de esta función y recibirá 3 parámetros esenciales. Primero **regexExpresion** que será una expresión regular a buscar, **line** será la cadena donde se buscarán expresiones regulares y **newString** será la nueva cadena que sustituirá a todas las coincidencias de **regexExpresion** en **line**. El Código 3.13 es la implementación.

```
1  # Funcion para buscar coincidencias de una expresion regular
2  # en una cadena
3  def replaceRegexWith(self, line, regexExpresion, newString):
4      lineOK = line
5      for match in re.finditer(regexExpresion, line, re.
6  UNICODE):
7          s = match.start()
8          e = match.end()
9          toReplace = line[s:e]
10         lineOK = lineOK.replace(toReplace, newString)
11     return lineOK
```

Código 3.13: Funcion para buscar expresion regular.

Para sustituir las expresiones regulares usaremos el Código 3.14

```
1  # Sustituye el regex por TAG en una cadena
2  def changeInLineRegexbyTag(self, line, regex, TAG):
3      lineOK = self.replaceRegexWith(line, regex, TAG)
4      return lineOK
```

Código 3.14: Función para sustituir expresion regular.

El Código 3.15 crea el diccionario que nos ayudara para varias clases donde se deben buscar stopwords. Con **open** abrimos y decodificamos el archivo indicado por **fileDiccionario** y se lee cada linea dividida por una coma ya que es un archivo csv, después las guardamos en un array llamado **myDiccionario** de esta forma ya lo tendremos disponible.

```
1  # crea un diccionario dado previamente el nombre del archivo
   # que contiene el diccionario
```

```

2     def createDictionary(self):
3         f = open(self.fileDiccionario, encoding='utf-8')
4         lines = f.readlines()
5         self.miDiccionario = np.array([])
6         for line in lines:
7             x = line.split(",")
8             self.miDiccionario=np.append(self.miDiccionario,np.
array([[x[0],x[1],x[2],x[3]]]))
9         self.miDiccionario=self.miDiccionario.reshape(-1,4)

```

Código 3.15: Creación del diccionario.

El archivo de stopwords contiene saltos de línea y estos pueden afectar en cuanto a eficiencia. Para remover estos en caso de aparecer lo harán en forma textual, por esto con **str.replace** bastara para remplazarlos por un carácter vacío, después nos quedaremos solo con los stopwords, así que solo se agregan a la lista **misStopWords** como lo muestra el Código 3.16.

```

1     # crea un diccionario dado previamente el nombre del archivo
que contiene el diccionario
2     def loadStopSwordsFromFile(self):
3         self.misStopWords= []
4         with open(self.fileStopWords, encoding='latin-1') as f:
5             for line in f:
6                 line = line.replace("\n","")
7                 self.misStopWords.append(line)

```

Código 3.16: Elimina los saltos de línea del archivo y guarda las stopwords.

Ya contamos con algunos stopwords en una lista, misma que podemos usar en el Código 3.17 para hacer consultas y retornando un valor booleano verificamos si una palabra esta en esa lista o no, análogo a esto ya sabremos si esta palabra es una stopword.

```

1     def isStopWordFile(self, word):
2         for dicWord in self.misStopWords:
3             if word == dicWord:
4                 return True
5         return False

```

Código 3.17: Verificación palabras en la lista de stopwords.

Remover palabras vacías (*stopwords*)

Las palabras vacías o palabras comunes (*stopwords* es el término usado en inglés en el área de procesamiento de lenguaje natural) se usan meramente para darle sentido a un texto, ya que normalmente son artículos, preposiciones, etc. Realmente no son útiles para aplicar un análisis de sentimientos porque no se extrae mucha información de este tipo de palabras sobre una oración, así que se puede decir que estorbarían si se incluyen.

Ejemplo:

Texto con palabras vacías:

"Tomé muchas decisiones tontas en mi vida pero esta fue la más peligrosa y grave. Esto es lo que reconoce Abderrahmane Fadil un hombre que optó por no vacunarse y acabó hospitalizado en Reino Unido con COVID19."

Texto sin palabras vacías:

"Tomé decisiones tontas vida más peligrosa grave. Esto reconoce Abderrahmane Fadil hombre optó no vacunarse acabó hospitalizado Reino Unido COVID19"

Como se puede ver el segundo texto (sin palabras vacías) parece tiene poco sentido, pero si prestamos atención a las palabras clave se nota que esta persona estuvo enferma de gravedad y fue hospitalizado, por esto se puede inferir que es una mala noticia. Las stopwords casi siempre sobran en este tipo de análisis por esto a continuación crearemos una clase que se encargue de removerlas.

Esta clase se llamará `StopWordsRemover`, esta clase será capaz de eliminar las stopwords usando un archivo como diccionario. Esta clase forma Heredera directamente de la clase del Código 3.10 donde ya se crearon algunas funciones que serán útiles en este punto. Para evitar errores la variable booleana **`stopwordsReady`** tendrá como función saber si el diccionario ya fue cargado. Dicha implementación se visualiza en el Código 3.18.

```
1 import string
2 from PreProcesamientoGeneral import PreProcesamientoGeneral
3
4 class StopWordsRemover(PreProcesamientoGeneral):
```

```

5  'Elimina stopwords, digitos y signos de puntuaci n'
6  stopwordsReady = False

```

Código 3.18: Clase para remover stopwords

A continuación se usara un archivo como diccionario para remover las stopwords, primero se verificara que dicho diccionario ya esta disponible, en caso de no ser así, lo cargaremos desde esta clase. Después procedemos a dividir la cadena **line** para separar palabra por palabra, así solo necesitamos recorrer esta cadena en un ciclo por cada palabra haremos uso de la función implementada en el Código 3.17, este método nos confirmara si alguna palabra de esta cadena se encuentra en el diccionario de stopwords, en caso de no estar en el archivo se agregan a la variable **lineOK** para retornarlas separadas por comas. Véase el Código 3.25.

```

1  # Elimina stopwords usando un archivo diccionario
2  def removeStopWordsFile(self, line, file):
3      if not self.stopwordsReady:
4          self.setFileStopWords(file)
5          self.loadStopSwordsFromFile()
6          self.stopwordsReady = True
7      tempo = line.split()
8      lineOK = []
9      for w in tempo:
10         if not self.isStopWordFile(w):
11             lineOK.append(w)
12     return " ".join(lineOK)

```

Código 3.19: Remueve stopwords desde el diccionario de stopwords

Se pueden remover signos de puntuación con ayuda de los módulos **str**, **line** y **string** que nos permiten en conjunto hacer una transformación y sustituir los signos de puntuación de **string** mas los signos "¡¿", por caracteres vacíos y después revertir la transformación para guardar los cambios. Esta implementación se encuentra en el Código 3.20.

```

1  #Elimina los signos de puntuaci n de una cadena
2  def removePunctuationInLine(self,line):
3      translatorPunct = str.maketrans('',' ', string.
4      punctuation + chr(191) + chr(161))
5      line = line.translate(translatorPunct)
6      return line

```

Código 3.20: Remueve los signos de puntuación de una cadena

Cambiar tildes

Las tildes o acentos normalmente pueden generar conflictos cuando son decodificados o leídos por un compilador, por esta razón se decidió prescindir de estos. El hecho remover esto no significa que la palabra se quedará sin la vocal que tiene el acento, solo se modificara para que sea remplazada por la misma letra sin acento. Se diseño una clase especifica para remplazar cada vocal con acentos de manera simple usando el método **replace** de la cadena y colocando dos parámetros, la letra con acento y la letra por cual sera remplazada respectivamente, esta clase sera llamada **AccentsChanger**. El Código 3.21 muestra la implementación de dicha clase.

```
1 class AccentsChanger():
2     'Clase para eliminar acentos y comillas'
3     # =====
4     # Sustituye los acentos por la vocal sin acento
5     def removeAccents(self,line):
```

Código 3.21: Remueve los acentos en una cadena.

Remover nombres de usuario y URL's

Los *Username* o *nombres de usuario* son ID que se distinguen por comenzar con un "@" seguido del nombre de usuario, estos pasan a segundo plano ya que en este trabajo de tesis no es necesario saber quien escribió el tweet para aplicar un análisis de sentimientos. No es necesario remover los nombres pero si es conveniente convertirlos en un especie de anónimo sustituyéndolos por una cadena **USERNAME** así sólo se hace uso de una expresión regular para nombres de usuario. Estos como ya sabemos tienen la forma "@Username_123" como vemos pueden contener números y símbolos. Esta clase dedicada a esto se llamara **UserNamesURL** y esta hereda de la clase **PreProcesamientoGeneral** por ello podemos usar el método

changeInLineRegexbyTag así sólo indicamos la cadena donde se realizaran los cambios, la expresión regular a buscar y el TAG este sera la nueva cadena. El Código 3.22 muestra a implementación de esta clase.

```
1 import re
2 from PreProcesamientoGeneral import PreProcesamientoGeneral
3
4 s"Esta clase est dise ada para sustituir los nombres de
   usuario y los url"
5 class UserNamesURL(PreProcesamientoGeneral):
6     # =====
7     # Sustituye los nombres de usuario por la palabra TAG, que
   puede USERNAME.
8     # Los nombres de usuario comienzan con @
9     def changeUsername(self, line, TAG):
10         return self.changeInLineRegexbyTag(line, '@\w+[^s]',
   TAG)
```

Código 3.22: Remueve los nombres de usuarios.

Ahora un *URL* es una cadena de texto que hace referencia a un vídeo, imagen, noticia, foro o blog de algún sitio web, estas cadenas no aportan nada a un análisis sentimental la mayoría de los tweets contienen este detalles por el hecho de hacer referencia al sitio de donde se compartió el contenido del tweet. Una vez que el tweet fue recolectado. Además de lo anterior las URL's pueden hacer referencia a la mismo sitio y ser diferentes cadenas esto nos afecta en la recolección de tweets en el Código 3.4, por ello se implemento la clase **UserNamesURL** que tiene como propósito remover nombres de usuario y URL's.

La remoción de URL's también sera realizada con el uso de expresiones regulares en el Código 3.23 de la clase **UserNamesURL**. Por su estructura es sencillo generar una expresión para identificar los URL en una cadena. Usando la biblioteca **re** podemos substituir una expresión regular por una cadena. En esta tesis se trabaja sobre un archivo csv y en ocasiones cuando una expresión se encuentra al final de la cadena se puede perder el apostrofo que indican el fin del texto por esto se verifica que la cadena aun lo contenga y si esta no lo tiene habrá que agregarlo.

```
1 def changeURL(self, line, TAG):
```

```

2     line = str(line)
3     lineOK = re.sub(r'^https?:\\/\\/.*[\\r\\n]*', TAG, line,
4     flags=re.MULTILINE)
5     lineOK = re.sub(r"http\\S+", TAG, lineOK)
6     if lineOK[ len(lineOK) - 1 ] != "''" :
7         lineOK = lineOK + "''"
8     return lineOK

```

Código 3.23: Remueve los URL en una cadena

Preprocesamiento

Hasta este punto la clase **PreProcesamientoGeneral** (Código 3.10) nos ayudará en eficiencia para clases que tendrán un fin específico como remover puntuaciones y stopwords. Aun así se necesita crear una clase para cargar los datos necesarios de estas clases. El objeto **preprocesamiento** podrá usar todas las funciones y atributos que tengan estas clases. De esta forma todo lo que concierne a minería de texto estaría concentrado en un solo objeto.

Como se aprecia en el Código 3.24 la clase **preprocesamiento** esta diseñada para usar todos los métodos y clases antes implementadas. Para poder usarlas hay que crear objetos miembro de estas clases en el constructor, estas son **stopWordsRemover** (Código 3.18), **AccentsChanger** (Código 3.21) y **UserNamesURL** (Código 3.22).

```

1 import pandas as pd
2 from AccentsChanger import AccentsChanger
3 from UserNamesURL import UserNamesURL
4 from StopWordsRemover import StopWordsRemover
5
6 class Preprocesamiento:
7     'Clase para preprocesar archivos de texto'
8     def __init__(self):
9         self.stopWordsRemover = StopWordsRemover()
10        self.UserNamesURL = UserNamesURL()
11        self.accentsChanger = AccentsChanger()

```

Código 3.24: Clase de procesamiento previo para archivos de texto.

En cuanto a stopwords, **removeStopWordsFile** puede hacer uso su método homónimo de la clase **stopwordsRemover** para eliminar estas palabras mediante la implementación de un ciclo que tome todo el conjunto de datos y los use como una

lista para leer cada línea del archivo, es importante recordar que cada línea de este archivo es un tweet por eso debemos hacerlo línea por línea. Por último retornamos una **Serie** de pandas con el tweet sin stopwords. En el Código 3.25 podemos ver como se implementa lo anterior.

```
1      # Elimina stopwords usando un archivo
2      def removeStopWordsFile(self, lista, fileDictionary):
3          nuevo = []
4          for line in lista:
5              line = str(line)
6              nuevo.append(self.stopWordsRemover.
7      removeStopWordsFile(line, fileDictionary))
          return pd.Series(nuevo)
```

Código 3.25: Elimina stopwords desde a clase preprocesamiento.

En el Código 3.26 se puede ver que proceso para remover acentos, URL's, nombres de usuario y puntuaciones desde esta clase **Preprocesamiento** es similar al de stopwords que anteriormente se implemento (Código 3.25), pues de la misma forma se crea un método, en este caso **removeAccents**, que pueda leer cada línea para enviarla a la clase **AccentsChanger** y remover estos con el método necesario. Después guardarlos en una Serie de pandas para retornarlos sin estos detalles.

```
1      # Remeve acentos palabra por palabra desde la clase
2      accentsChanger
3      def removeAccents(self, lista):
4          nuevo = []
5          for line in lista:
6              line = str(line)
7              nuevo.append(self.accentsChanger.removeAccents(line)
          )
          return pd.Series(nuevo)
```

Código 3.26: Elimina acentos desde a clase preprocesamiento.

Siguiendo esta forma podemos hacerlo así remover:

- **URLs** (Código 3.27).
- **Nombres de usuario** (Código 3.28).
- **Signos de puntuación** (Código 3.29).

```

1  def changeURLs(self, lista, URL):
2      nuevo = []
3      for line in lista:
4          line = str(line)
5          nuevo.append(self.UserNamesURL.changeURL(line, URL))
6      return pd.Series(nuevo)

```

Código 3.27: Cambia las URL desde a clase preprocesamiento.

```

1  def changeUserNames(self, lista, USERNAME):
2      nuevo = []
3      for line in lista:
4          line = str(line)
5          nuevo.append(self.UserNamesURL.changeUsername(line,
6      USERNAME))
7      return pd.Series(nuevo)

```

Código 3.28: Cambia nombres de usuario desde a clase preprocesamiento.

```

1  def changeURLs(self, lista, URL):
2      nuevo = []
3      for line in lista:
4          line = str(line)
5          nuevo.append(self.UserNamesURL.changeURL(line, URL))
6      return pd.Series(nuevo)

```

Código 3.29: Cambia signos de puntutacion desde a clase preprocesamiento.

De ahora en adelante ya esta lista la clase para hacer uso de las clases secundarias, pero aun debemos crear un par de métodos para cargar y leer el archivo csv de datos con el que se trabajara en esta tesis mismo archivo que se creo en el Código 3.9 y se etiqueto en la Sección 3.3.

Para cargar el archivo en bastara con crear un setter llamado **setFile** para cargar la ruta de dicho archivo.

```

1  # Carga el nombre del archivo de datos
2  def setFile(self, fileName):
3      self.fileName = fileName

```

Código 3.30: Carga ruta del archivo de datos.

En esta tesis se trabajo solamente sobre archivos csv, pero también se hicieron pruebas con archivos xlsx, estas resultaron negativas pues la codificación generaba

bastantes problemas en lectura pero igual es una vía alterna al uso de csv. En el Código 3.31 se puede apreciar que la forma de leer del método **readColumn** es por columnas ya que en una se encuentra el contenido del tweet y en la siguiente esta la etiqueta de sentimiento del tweet. En el código del método primero verificamos la extensión del archivo para elegir la forma de lectura con la biblioteca **pandas**, podemos leer varios tipos de archivos en python con esta biblioteca, en este caso solo usaremos dos métodos de lectura **read_csv** y **read_excel** para csv y xlsx respectivamente. Una vez leído todo el archivo solo se retorna la columna proporcionada en los parámetros.

```
1  # Lee una columna del archivo segun su extension csv o xlsx
2  def readColumn(self, column):
3      if '.csv' in self.fileName:
4          x = pd.read_csv(self.fileName)
5      elif '.xlsx' in self.fileName:
6          x = pd.read_excel(self.fileName)
7      return x.iloc[:, column]
```

Código 3.31: Lectura del archivo de datos.

3.4.1 Diagrama UML de clase detallado de la arquitectura

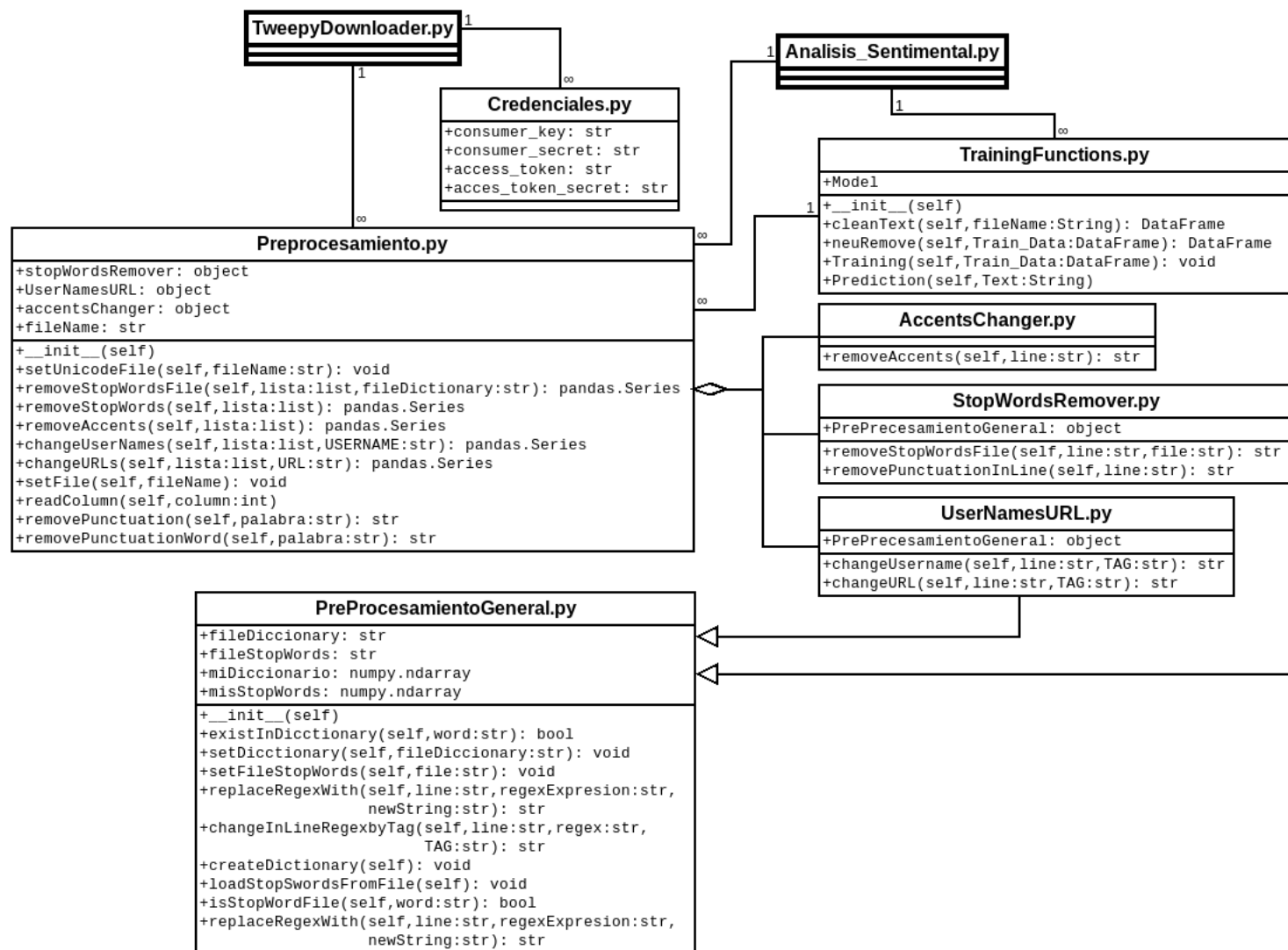


Figura 3.6: Diagrama de clases para el pre procesamiento de documentos

3.5 Proceso de análisis y entrenamiento

Después de haber desarrollado los métodos necesarios para llevar a cabo una recolección y un procesamiento previo eficiente a los datos, con el objetivo de tener datos limpios y útiles, para realizar un análisis efectivo complementamos la dinámica de este trabajo de tesis con llevar a cabo el análisis de sentimientos y el entrenamiento del modelo de predicción.

En el preprocesamiento de datos que se especifico a lo largo de la Sección 3.4, la clase comprendía la remoción de detalles que no aportan nada al análisis y funciones para la lectura del conjunto de datos, este es la primer implementación e instancia que debemos hacer.

Es importante considerar que la clase **preprocesamiento** será implementada en esta etapa ya que contribuye en la etapa de NLP de forma importante. Antes de comenzar a con procesos de entrenamiento y extracción de información con el uso de técnicas de machine learning y aprendizaje supervisado. Se deben tratar los datos por ello comenzaremos con este punto y una vez que este completo daremos inicio a la implementación de los procesos de análisis y entrenamiento.

3.5.1 Análisis de sentimientos

Principalmente en el análisis sentimental eficiente debe haber un tratamiento de datos igual de eficiente por ello creamos un solo método que pueda implementar todo el preprocesamiento ya definido en dicha clase de la Sección 3.4, este sera el primero de varios métodos en una nueva clase llamada **ModelFunctions**. La clase se encargara no sólo de proporcionar el método que realizara todo el text mining del preprocesamiento, sino que también implementara métodos para el llevar a cabo el entrenamiento y pruebas del modelo.

Para comenzar con esta clase, primero definimos las bibliotecas necesarias para los métodos que serán desarrollados, el Código 3.32 contiene las bibliotecas que serán

imprescindibles para el modelo.

```
1 import spacy
2 import string
3 import numpy as np
4 import pandas as pd
5
6 from Preprocesamiento import Preprocesamiento
7 from sklearn.feature_extraction.text import TfidfVectorizer
8 from sklearn.pipeline import Pipeline
9 from sklearn.model_selection import train_test_split
10 from sklearn.metrics import classification_report,
    confusion_matrix
11 from sklearn.model_selection import cross_validate
12 from sklearn.svm import LinearSVC
13 from spacy.lang.es.stop_words import STOP_WORDS
```

Código 3.32: Bibliotecas para métodos de la clase **ModelFunctions**.

Cada una de estas bibliotecas es importante ya que los métodos se apoyan de spacy, sklearn, numpy, pandas, string y clases ya desarrolladas en este trabajo. Como comentamos anteriormente definiremos la clase como **ModelFunctions** y crearemos un constructor para dicha clase inicializando un atributo miembro llamado **Model** que por ahora contendrá un cadena vacía pero mas adelante sera de vital importancia porque albergara el modelo una vez entrenado. El Código 3.33 muestra como crear este constructor.

```
1 class ModelFunctions:
2
3     #Constructor
4     def __init__(self):
5         self.Model = ''
```

Código 3.33: Constructor de la clase **ModelFunctions**.

La implementación del método para realizar todo el preprocesamiento se llamará **cleanText**, este método necesita un parámetro imprescindible del conjunto de datos con el que se trabajará, recibirá una cadena llamada **fileName** y esta es de suma importancia por que se tratara de la ruta donde se encuentra el conjunto de datos (path) para entrenamiento y pruebas. Hacer esto nos facilita consumir el método de la clase **preprocesamiento** llamado **setFile** ya que solo es necesario pasar como parámetro

la ruta para cargar el archivo al modelo. Pero antes de usar cualquier método de la clase **preprocesamiento** definida en el Código 3.24 necesitamos instanciar un objeto de la clase en primer lugar, llamaremos **p** a esta instancia, ahora mediante **p** podemos hacer uso de este y otros métodos para llevar a cabo procesos importantes utilizados para realizar la limpieza de texto. En el Código 3.34 podemos ver la definicion del path para el archivo que se cargara en el método **setFile** pasando la cadena **fileName** como parámetro.

```
1 def cleanText(self, fileName):
2     print("\n\n
3     *****\n\nIniciando preprocesamiento
4     de:", fileName)
5     p = Preprocesamiento()
    p.setFile(fileName)
```

Código 3.34: Método cleanText e ingreso de archivo de preprocesamiento.

Después de haber ejecutado el método **setFile** ya se encontraría cargado el archivo del conjunto de datos en el modelo, para ser específicos en el objeto **p** de la clase de **preprocesamiento**, estos datos fueron etiquetados manualmente en un csv, es decir, el contenido y el sentimiento del tweet se encuentran separados por comas y en este momento no requerimos de la etiqueta del sentimiento solo trataremos el texto, es necesario separar el contenido y el sentimiento para hacer esto el objeto **p** contiene el método **readColoum** que tal y como su traducción lo indica lee una columna del archivo ya cargado simplemente indicando el índice de la columna que se extraerá, este metodo permite leer de archivos csv ó xlsx y retorna los valores como una lista unidimensional. Esto debido a que los set de datos salida que proporciona el API de Twitter consumida en la Sección 3.2.2 exporta un archivo de datos separados por comas (.csv) en este trabajo se enfocara en datos almacenados en este formato. En el archivo los datos se encuentran almacenados de la siguiente forma: [*"Contenido del tweet", "etiqueta de sentimiento"*] así que podemos separar los contenido de los tweets y etiquetas en dos listas distintas llamadas **texto** y **textoSnt** respectivamente como el Código 3.35 lo muestra.

```

1      texto = p.readColumn(0) # Supone que el contenido est
    en la columna 0
2      textoSnt = p.readColumn(1) # Supone que el sentimiento
    est en la columna 1

```

Código 3.35: Separación de contenido de set de datos.

Una vez que hemos separado el contenido del conjunto de datos y la lista **texto** del Código 3.35, contiene solo el texto del tweet podemos iniciar con el procesamiento de los datos de entrenamiento con los métodos desarrollados en la sección 3.4 dichos métodos trabajan sobre una lista unidimensional y cada uno de los elementos contenidos, es decir, el método se aplicara a toda la lista.

Como primer paso en el preprocesamiento o limpieza de texto en esta tesis, el objeto P permite deshacernos de los nombres de usuario, estos son algo que no aporta nada al análisis e incluso puede llegar a entorpecer el entrenamiento si no se tratan, el método **changeUserNames** hace prácticamente lo mismo que el método **changeURL** utilizado para homologar ligas de Internet en el Código 3.23. Debido a que los nombres de usuario en Twitter tienen una estructura sencilla y fácil de identificar, el método **changeUserNames** realizará la homologación de las cadenas en el contenido de los tweets que encajen con la expresión regular que identifica nombres de usuario, así todos los nombres se sustituirán en los contenidos de los tweets por la cadena que coloquemos como parámetro, en este caso será la cadena "USERNAME", el Código 3.36 muestra el consumo de este método.

```

1      texto = p.changeUserNames(texto, "USERNAME")

```

Código 3.36: Homologación de nombres de usuario en tweets.

Saber si un tweet contiene signos de puntuación es un punto muy importante en este trabajo de tesis, este detalle puede generar errores en la etapa de NLP ya que estos signos solo dan coherencia a una frase pero no son de utilidad para extraer sentimientos, teniendo esto presente el método **removePunctuation** quitara del tweet comas, puntos, punto y coma, comillas, signos de exclamación y pregunta, etc.

Este método utiliza un diccionario de signos del modulo **String** de Python para hacer esto el Código 3.37 muestra como se lleva acabo la invocación del método **removePunctuation**.

Ya habíamos comentado que son las stopwords en la sección 3.4 y también el motivo por el que se deben remover. Básicamente estas palabras al igual que los signos de puntuación solo generan ruido en el análisis sentimental. El método **removeStopwords** del objeto **p**, utiliza como diccionario un lista de stopwords en texto plano , el método recibe el path del este archivo como parámetro de esta forma internamente en el método usa el contenido del archivo para identificar stopwords en el cada tweet de la lista **texto** y al encontrar coincidencias se remueven.

Por motivos de codificación en los comentarios de los códigos no se utilizan tildes en las palabras para evitar errores. Por esta misma razón también los debemos remover del conjunto de datos ya que son detectados como un texto UNICODE y generamos inconvenientes en el análisis, el método **removeAccents** sustituye las vocales con tilde por la misma vocal sin tilde así podemos estar seguros que las palabras serán integras en el proceso de NLP. El Código 3.37 implementa este método.

```
1      texto = p.removePunctuation(texto)
2      texto = p.removeStopWordsFile(texto, "Dictionaries/
stopwords.txt")
3      texto = p.removeAccents(texto)
```

Código 3.37: Remoción de signos de puntuación y stopwords del texto de los tweets.

Ahora el método **textClean** puede dar preprocesamiento a cualquier conjunto de datos, cuando comenzamos con la implementación del método separamos el conjunto de datos en dos listas diferentes llamadas **texto** y **textoSnt**, uno alberga la lista de tweets y el otro almacena el sentimiento de estos tweets respectivamente. Bien, para concluir con el método debemos volver a hacerlos una sola lista bi dimencional. Para esto debemos utilizar la librería **pandas** para incrustarlos en un DataFrame llamado **textoM** que tendrá definidas sus columnas con los encabezados "*content*" y "*sentiment*" que ayudarán a facilitar el acceso a su contenido. Así solo resta asignar a

la columna "content" la lista **texto** y a la columna "sentiment" la lista **textoSnt** de etiquetas de sentimientos, una vez realizado este proceso solo resta retornar el DataFrame con los tweets casi listos para ser analizados. Este proceso se puede ver en el Código 3.38;

```
1      textoM = pd.DataFrame(columns=['content', 'sentiment'])
2      textoM.content = texto
3      textoM.sentiment = textoSnt
4      print("\n\n
5      *****
6      + "*****\n\nPreprocesamiento listo
      !!!.")
      return textoM
```

Código 3.38: Reintegración de contenedores de datos

REMOVED TWEETS NEUTRALES.

El modelo de predicción trabajado para esta tesis es de clasificación binaria, es decir se trata de indicar un sentimiento positivo o negativo. Por esta razón, los tweets etiquetados como neutrales en el conjunto de datos deben ser removidos del set de entrenamiento, ya que solo generan confusión al modelo.

Pero no serán removidos del conjunto de datos por que son un buen parámetro para la etapa de test o pruebas dado que es interesante conocer el veredicto del modelo en estos casos. Remover estos tweets es relativamente sencillo pues ya que el conjunto de datos que retorna la función **textClean** es un DataFrame y podemos indicar a Python con el método de DataFrame **index** que recupere los índices de los elementos donde la columna de sentimiento contenga la etiqueta "neu" que indica que es un tweet donde no fue posible asignar un sentimiento, contando con los índices de estos tweets en **Train_Data** serán removidos fácilmente con el método **drop** del DataFrame. El Código 3.39 es la implementación lo que hemos estado comentando sobre los tweets neutrales definido en un método llamado **neuRemove** que recibe un set de datos del tipo DataFrame con el que realiza el filtro de tweets y retorna el mismo DataFrame pero con el filtro aplicado.

```

1     def neuRemove(self, Train_Data):
2         indexNames = Train_Data[(Train_Data['sentiment'] == 'neu
3         ')] .index
4         Train_Data.drop(indexNames, inplace=True)
5         return Train_Data

```

Código 3.39: Remoción de tweets neutrales

3.5.2 Entrenamiento

A pesar de que en la sección anterior ya tratamos la integridad de los datos para el entrenamiento aun no se ha comenzado a realizar nada de NLP, en esta sección comenzaremos a realizar la parte angular de este trabajo de tesis, el análisis sentimental en el que se verá aplicado en el entrenamiento que realizaremos al modelo. Todo este proceso se verá implementado en un solo método llamado **Training** y recibirá como parámetro un conjunto de datos para realizar el entrenamiento, en este caso sera el conjunto de tweets filtrados en el Código 3.39,

Para comenzar a analizar los datos en el Código 3.41 se creó un método llamado **myTokenizerCleaner** este método contiene un método anidado que consume el modelo de procesamiento de lenguaje natural de **Spacy** mismo que definimos como **nlp** que en resumen tomará cada tweet y lo se encargara de convertirlo en un documento de spacy, aplicando un pipeline convirtiendo cada palabra del tweet en un token con ciertas características como un **part of speech** y un **lemma** con los cuales podremos saber que parte de la oración ocupa y cual es la palabra canónica respectivamente, con estos datos se puede realizar filtro más para validar que solo estamos utilizando tokens útiles, como ya lo hemos comentado anteriormente los tokens que contengan la etiqueta *"-PRON-"* indican que se trata de un pronombre y estos no aportan información sobre el contenido del tweet, estas partes del tweet son desechadas, por otro lado además de validar que estas palabras o tokens no sean pronombres se utiliza el diccionario de stopwords de spacy (**stopwords.spacy**) para hacer un filtro de stopwords con el fin de tener la mínima probabilidad de que alguna palabra vacía se utilice en el entrenamiento y predicciones del modelo así como

también los signos de puntuación, los tokens útiles son retornados en el contenedor **clean_tokens**, los contenedores **puntuacion** y **stopwords_spacy** son globales y se definen en el Código 3.40, se concatenan los códigos *chr(191) = ¿* y *chr(161) = ¡* debido a que el diccionario de puntuación se basa en el idioma ingles donde estos signos no se usan.

```
1 stopwords_spacy = list(STOP_WORDS)
2 puntuacion = string.punctuation + chr(191) + chr(161)
```

Código 3.40: Contenedores de stopwords de Spacy y signos de puntuación

```

1  def Training(self, Train_Data):
2      print("\n\n*****")
3          + "*****\n\nIniciando
entrenamiento!!!")
4      def myTokenizerCleaner(sentence):
5          doc = nlp(sentence)
6          tokens=[]
7          for token in doc:
8              if token.pos_ != "PRON":
9                  temp = token.lemma_.strip()
10                     tokens.append(temp)
11             clean_tokens = []
12             for token in tokens:
13                 if token not in stopwords_spacy and token not in
puntuacion:
14                     clean_tokens.append(token)
15             return clean_tokens

```

Código 3.41: Definición del metodo `myTokenizerCleaner`

El análisis de estas palabras y la metadata que logramos obtener del NLP de Spacy es vital dado que la composición de las palabras nos indicaran un sentimiento positivo o negativo, para canalizar esta información extraída y definir el sentimiento que refleja el tweet debemos hacer posible su clasificación. Nos apoyamos de la librería **sklearn** que cuenta con el método **TfidfVectorizer** que implementa la técnica **Frecuencia de término - Frecuencia inversa de documento (TF-IDF)** este método convertirá cada tweet que tokenizemos a una matriz o vector de valores de TF-IDF que indica la frecuencia de una palabra, así como también la escasez de otras que se encuentren en el contenido. Adicionalmente es posible indicarle el método que realizara el análisis de

tokens en este caso ya contamos con uno se trata de **myTokenizerCleaner** y basta pasarlo como parámetro **tokenizer** y para hacer referencia al objeto que nos devolverá utilizaremos la variable **tfidf**.

Para poder clasificar estas matrices implementamos una **Máquina de Vectores de Soporte (MVS)** haciendo consumo de **LinearSVC** de **sklearn** para proporcionarnos una MVS llamada **classifier**,

Ahora que ya contamos con un método para vectorizado de tweets (**tfidf**) y un clasificador (**classifier**), **sklearn** nos permite crear nuestro **Pipeline** indicando la secuencia de los métodos de procesamiento que aplicara el modelo para su entrenamiento como para testeo al solicitar una predicción. En este caso nuestro Pipeline integrara inicialmente de **tfidf** y **classifier** de esta forma primero se realizara el análisis de NLP y vectorizaran los datos (*'tdif'*) y después se clasificaran (*'clf'*) con una maquina de vectores de soporte. El Código 3.42 ilustra la implementación y construcción del pipeline.

```
1      tfidf = TfidfVectorizer(tokenizer = myTokenizerCleaner)
2      classifier = LinearSVC()
3      clf = Pipeline([('tdif', tfidf), ('clf', classifier)])
```

Código 3.42: Construcción de pipeline del modelo.

El aprendizaje supervisado implica que los datos que se usan en el entrenamiento de un modelo deben estar previamente etiquetados, en este caso el **DataFrame Train_Data** contiene el texto del tweet y su respectiva etiqueta asignada. Para entrenar el modelo de predicción es necesario dividir el set de datos con el propósito de obtener cifras de rendimiento, es decir, podríamos utilizar el 100% del contenido de set de datos para entrenar el modelo, ya que mientras mas datos tenga para aprender mas precisa serian sus predicciones pero no tendríamos un set de testeo para calcular estadísticas de rendimiento. Para atender esta necesidad en el Código 3.43 nos apoyamos nuevamente de **sklearn** para consumir el método **train_test_split** que nos ayudara a dividir aleatoriamente el set de datos contenido en **Train_Data** en un 80% para entrenamiento y un 20% para pruebas. Para evitar que la división de datos

siempre sea siempre la misma cada vez que se ejecuta el barajado pasamos un entero aleatorio en el parámetro **random_state**; mas adelante explicaremos porque es conveniente que la división de datos sea diferente en cada barajado.

```
1 X_train, X_test, y_train, y_test = train_test_split(  
  Train_Data['content'],  
2  Train_Data['sentiment'],  
3  test_size = 0.2,  
4  random_state = np.random.randint(0, 100, size=1)[0])
```

Código 3.43: Definición de set de datos para entrenamiento y test

Necesitaremos convertir los contenedores de objetos **y_train** y **y_test** a contenedores de enteros ya que requerimos de ellos para realizar cálculos de estadísticas como *F-Score*, *recall* y *precisión*.

Ahora que ya contamos con los conjuntos de datos completamente ordenados para cada propósito, daremos paso a entrenar y extraer las métricas de rendimiento del modelo como lo muestra el Código 3.44. El objeto **clf** por medio de su método **fit** permite recibir el conjunto de tweets y su respectiva etiqueta. Este método como su nombre lo dice entrenara modelo y obtendremos una maquina de vectores de soporte como modelo predictivo. Para poder hacer uso de las predicciones se utiliza el método **predict**, para consumirlo es necesario indicarle un objeto iterable con datos para realizar predicciones en nuestro caso sera el objeto **x_test** que cuenta con el 20% de los tweets destinado para testeo, haremos referencia a las predicciones de testeo como **y_pred**. Ahora ya contamos con **y_test** y **y_pred** donde contamos con las etiquetas correctas y las etiquetas asignadas por el modelo. Con esta información es posible realizar los cálculos de rendimiento antes mencionados, por suerte **sklearn** implementa un forma rápida de obtener a partir de los resultados de predicciones y testeo. El método **classification_report** a partir de los resultados obtenidos al comparar los conjuntos iterables de testeo y predicciones del modelo construye una tabla con los valores alcanzados en **Precisión**, **Recall**, **F1-score** y **Support**. La

Tabla 3.1 muestra un ejemplo del uso de este método.

```
1 y_train, y_test = y_train.astype('int'), y_test.  
  astype('int')  
2 clf.fit(X_train, y_train)  
3 y_pred = clf.predict(X_test)  
4 report = classification_report(y_test, y_pred,  
  output_dict=True)
```

Código 3.44: Entrenamiento y calculo de métricas del SVM

Este módulo de **sklearn** nos proporciona un dato muy importante nos referimos a la **precisión (accuracy)** al cual accedemos a través de **report** con el índice *"accuracy"*, este parámetro nos indica la eficiencia específica del entrenamiento realizado, podemos utilizar este parámetro para establecer un valor mínimo para cada entrenamiento que se realice con el propósito de garantizar un entrenamiento óptimo de nuestro modelo. Así que indicaremos que la precisión tenga un valor mínimo de **0.8** para que sea aceptable para el modelo. El Código 3.45 muestra como realizar esta implementación.

```
1 while accuracy_score < 0.8:  
2     X_train, X_test, y_train, y_test = train_test_split(  
3     Train_Data['content'],  
4     Train_Data['sentiment'],  
5     test_size = 0.2,  
6     random_state = np.random.randint(0, 100, size=1)[0])  
7     y_train, y_test = y_train.astype('int'), y_test.  
8     astype('int')  
9     clf.fit(X_train, y_train)  
10    y_pred = clf.predict(X_test)  
    report = classification_report(y_test, y_pred,  
    output_dict=True)  
    accuracy_score = report['accuracy']
```

Código 3.45: Implementación de precisión mínima de la SVM

Estos datos obtenidos de las pruebas se utilizaran también para obtener otro aspecto importante de análisis cuando se implementa aprendizaje automático, se trata de la **matriz de confusión** básicamente esta métrica permite percibir los aciertos y errores en cada predicción efectuada en la fase de test, es decir cada prueba que el modelo realizo debe tener 1 de los 4 estados o términos que implementa la matriz de

Tabla 3.1: Promedio del desempeño de SVM

Clase	Precisión	Recall	F1-score	Support
0	0.85	0.92	0.88	36
1	0.83	0.71	0.77	21
accuracy			0.84	57
macro avg	0.84	0.82	0.82	57
weighted avg	0.84	0.84	0.84	57

confusión: *verdaderos positivos (VP)*, *falsos positivos (FP)*, *falsos negativos (FN)* y *verdadero negativos (VN)*. Para calcular esta métrica y conocer el rendimiento del modelo en el Código 3.46 utilizamos el módulo **confusion_matrix** de **sklearn** que al proporcionarle el set de test (**y_tests**) y el set de resultados (**y_pred**) devolverá los valores obtenidos por el modelo necesarios para construir nuestra matriz de confusión tal como lo muestra la tabla 3.2.

```
1 print("\n",classification_report(y_test, y_pred),"\n")  
2 tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()  
3  
4 print("Matriz de confusion: \n")  
5 print("\t\t-----\n")  
6 print(f "\t\tTP:{tp}\t\tFN:{fn}\t\t")  
7 print(f "\t\tFP:{fp}\t\tTN:{tn}\t\t")  
8 print("\t\t-----\n\n")
```

Código 3.46: Impresión de métricas del SVM

Como último paso en el Código 3.47 definimos como propiedad **Model** de la clase **ModeloFunctions** al modelo generado es decir el objeto **clf** así podemos hacer uso del modelo.

La clase **ModelFunctions** esta completa y lista para hacer uso del modelo predictivo como la Figura 3.7 nos indica al completar el preprocesamiento y entrenamiento.

En el próximo capítulo se abordará la fase de pruebas del modelo y se documentarán las pruebas aplicadas a este modelo de predicción de SVM.

```
1 print("Entrenamiento listo!!!\n\n*****")
```

```

2         + "
3         *****\n\n")
        self.Model = clf #Modelo con entrenamiento terminado

```

Código 3.47: Modelo con entrenamiento terminado.

		Condición Real	
		Positivo	Negativo
Predicción	Positivo	20	5
	Negativo	0	32

Tabla 3.2: Matriz de confusión obtenida del SVM.

```

*****
Iniciando preprocesamiento de: DataSets/DataSetTraining.csv

*****
Preprocesamiento listo!!!
|
*****

Iniciando entrenamiento!!!

              precision    recall  f1-score   support

     0       0.80      0.92      0.86         36
     1       0.81      0.62      0.70         21

 accuracy          0.81          0.81          0.81         57
 macro avg          0.81          0.77          0.78         57
weighted avg          0.81          0.81          0.80         57

Matriz de confusion:

+-----+
| TP:13 | | FN:8  | |
| FP:3  | | TN:33 | |
+-----+

Entrenamiento listo!!!
*****

```

Figura 3.7: Salida en terminal de Python al completar el preprocesamiento y entrenamiento.

Capítulo 4

PRUEBAS REALIZADAS

En este capítulo se presentan las pruebas realizadas a los modelos calibrados con la metodología explicada anteriormente.

La inmensa mayoría de trabajos sobre análisis de sentimientos con énfasis en ingeniería, tiende a mostrar como resultados finales los desempeños alcanzados por los modelos predictivos con el conjunto de datos de prueba. Sin embargo, en este trabajo, decidimos realizar algunas pruebas más, para explorar el alcance de los modelos calibrados.

Los resultados de desempeño de los modelos mostrados en el capítulo anterior, fueron obtenidos separando el conjunto de datos en un subconjunto de datos de prueba y otro de entrenamiento. Se decidió usar un conjunto de datos completamente diferente, pero relacionado al mismo tema de COVID19.

La metodología para realizar las predicciones consiste de las mismas tres etapas que desarrollamos en esta tesis:

- **Preprocesamiento:** Se cargará el conjunto de pruebas y se preprocesará.
- **Predicción:** Se usarán los modelos anteriores y se realizará la predicción.

En la fase de preprocesamiento haremos uso de los módulos desarrollados **Preprocesamiento** y **ModelFunctions**. Estos contienen métodos que diseñamos

para cargar y analizar contenido, así como también procesos para remover todo lo que en la fase de entrenamiento puede resultar irrelevante, como palabras vacías. Por otra parte, se utilizará el módulo **random** para poder realizar la selección aleatoria de datos (Ver Código 4.1).

```
1 from Preprocesamiento import Preprocesamiento
2 from ModelFunctions import ModelFunctions
3 import random as rd
```

Código 4.1: Importación de módulos para realizar pruebas.

Debido que el modelo que usaremos para pruebas es el mismo que calibramos en el capítulo de desarrollo omitiremos la fase de entrenamiento.

Ahora, como podemos ver en el Código 4.2 primero debemos definir las rutas de nuestros archivos con los conjuntos de datos para entrenamiento y pruebas.

```
1 trainingFile = "DataSets/DataSetTraining.csv"
2 testFile = "DataSets/DataSetTesting.csv"
```

Código 4.2: Método cleanText e ingreso de archivo de preprocesamiento.

Estas rutas son imprescindibles para esta tesis pero en este capítulo como prioridad tenemos a **testFile** ya que podemos considerar que contiene las pruebas mas complejas que podemos realizar. Dado que el desarrollo de esta tesis es modular podemos realizar pruebas simples en pocas lineas de código de Python. Es necesario para cargar el conjunto de datos de pruebas el modulo **preprocesamiento** ya que contamos con el métodos para cargar conjuntos de datos y lectura de su contenido como nos muestra el código 4.3.

```
1 preprocesamiento = Preprocesamiento()
2 preprocesamiento.setFile(testFile)
3 testText = preprocesamiento.readColumn(0)
```

Código 4.3: Carga y lectura de conjunto de pruebas.

El resultado de ejecutar estas lineas de código nos dará como resultado una colección de tweets en una **Serie** del modulo **pandas** a la cual llamamos **testText**.

En este trabajo de tesis se consideraron 10 muestras aleatorias como un número vasto de pruebas para obtener una percepción realista desempeño del modelo.

Con **testText** podemos recuperar un tweet del conjunto de datos de forma directa o de forma aleatoria, debido que nuestras pruebas se realizaran de forma aleatoria, el recuperado de tweets se hace mediante el modulo **random** que definimos como **rd** en las importaciones iniciales, obtendremos un valor entero aleatorio en el rango de 0 y el tamaño de la serie -1 con el método **randInt** y al texto que recuperemos con este índice dentro de **testText** lo colocaremos en el contenedor llamado **tweet**.

La instancia de **modelFunctions** nos permite acceder al modelo y realizar una predicción a través de **modelFunctions.Model.predict** e indicando a **tweet** como parámetro de siguiente forma: "[**tweet**]" para realizar el análisis y predicción, el resultado que obtendremos como ya lo hemos comentado se trata de un valor binario y lo almacenaremos en el contenedor **prediction**.

Ahora, contar con el contenido del tweet y el resultado proporcionado por el modelo nos permite comenzar a construir una cadena llamada **mensaje** que fungira como contenedor del tweet y su predicción. Esta cadena utilizara caracteres especiales para hacer mas estética la respuesta del modelo. A su vez utilizamos un operador ternario sobre el valor de **prediction** y concatenar un mensaje que indique si se trato de un tweet con un sentimiento negativo o positivo, dependiendo del resultado de la predicción almacenada y por ultimo imprimiremos el contenido de la cadena **mensaje** ya que esta completamente construida, en el Código 4.4.

```
1 for Renglon in range(0,10):
2     tweet = testText[rd.randint(0,len(testText) - 1)]
3     prediction = modelFunctions.Model.predict([tweet])
4     mensaje = tweet + "\n" + chr(9492) + "> Tweet " + ("POSITIVO " +
5     chr(128077) + "\n" if prediction == 1 else "NEGATIVO " + chr
    (128078) + "\n")
    print(mensaje)
```

Código 4.4: Ejecución e impresión de pruebas del modelo basado en SVM.

La ejecución de estas líneas de código de Python genero una salida en terminal como la que se muestra en la Figura 4.1.

```
'@MaykellTwin @BuenaFe_musica @rauldominguezb2 @Reylope13 @RobertoLaurenc8 @LeivaYosel @YamilkaOliva @JEAlvarezCuesta @Izamorano46
@Red26C @japortalmiranda #Cuba PRIMERA: _derrota militar al #Imperialismo en América (Girón 1961) _Eliminar la transmisión de
#Sida y sífilis de madre a hijo (2015) _En A. Latina en lograr control de la diabetes (2020) y vacuna vs COVID-19(Abdala) SINO ES
RECORD ES BUEN AVERAGE #PasionXCuba #CubaSalva https://t.co/P5TwhHPwpw'
↳ Tweet POSITIVO 👍

' España supera los 34 millones de personas con la pauta completa de vacunación frente a la #COVID19 El 926 % de la población
mayor de 40 años ya ha recibido la pauta completa de #VacunaCOVID19 #UnÉxitoColectivo #YoMeVacunoSeguro https://t.co/kikq0om2Wp'
↳ Tweet POSITIVO 👍

'#DatoDeLaVacuna: las vacunas contra el #COVID19 no cambian ni interactúan con tu ADN. Ni las vacunas de ARNm (Pfizer-BioNTech
[COMIRNATY] o Moderna) ni las de vector viral (J&J/Janssen) entran al núcleo de las células donde está el ADN. Más: https://
t.co/0bJ0o6nZMu https://t.co/AxAPr2aL6V'
↳ Tweet POSITIVO 👍

'Autoridades sanitarias de Bolivia identifican variante mu de coronavirus https://t.co/9pWZuis16y https://t.co/1FEuZ0nCp4'
↳ Tweet NEGATIVO 🚫

'El G20 se compromete a inmunizar a 40% de la población mundial en 2021 - https://t.co/jjzjW4oH6x #EVMadrid #G20 #vacunas
#covid19'
↳ Tweet POSITIVO 👍

'Arribaron 200 mil dosis de CanSino que serán aplicadas en poblaciones de difícil acceso https://t.co/FybqV4qEFZ #vacunas #covid
#coronavirus https://t.co/KRwx0w7Gqq'
↳ Tweet POSITIVO 👍

'Decimosexta rotación en ZONA ROJA del Centro Hospitalario para Contactos de Casos Confirmados en Pandemia #45GeneralísimoIPVCE.
Ahora reforzado con jóvenes de la COMUNIDAD. Al día de hoy se cumplen 215 días en el COMBATE a la COVID-19. #SomosContinuidad
#CubaMined. #FidelVive https://t.co/AhD6ZBijMe'
↳ Tweet NEGATIVO 🚫

'Lo que tenés que saber sobre la vacunación contra el coronavirus https://t.co/lnhw8pi7Ma https://t.co/xS7Idiidy'
↳ Tweet NEGATIVO 🚫

'Vacunas contra el COVID-19 durante el Desfile de El Salvador https://t.co/QloSWq3PGa'
↳ Tweet POSITIVO 👍

'Quién es la científica argentina que lidera desde el Reino Unido el desarrollo de fármacos contra el COVID-19 Por Valeria Román
@ValeriaRoman Medio: Infobae https://t.co/nGEoYnhMJN'
↳ Tweet NEGATIVO 🚫

In [119]: |
```

Figura 4.1: Resultado de las pruebas realizadas.

Como podemos ver en Figura 4.1 el modelo logro asignar a nuestro parecer 8 sentimientos correctamente y 2 incorrectos. A continuación explicaremos como asumimos algunos de estos resultados.

El modelo predijo un sentimiento positivo al tweet: *'#DatoDeLaVacuna: las vacunas contra el #COVID19 no cambian ni interactúan con tu ADN. Ni las vacunas de ARNm (Pfizer-BioNTech [COMIRNATY] o Moderna) ni las de vector viral (J&J/Janssen) entran al núcleo de las células donde está el ADN. Más: https://t.co/0bJ0o6nZMu https://t.co/AxAPr2aL6V'*, consideramos que su predicción fue correcta ya que este usuario compartió información verídica y positiva sobre las vacunas contra COVID-19, en su momento estas generaron dudas en personas que obtuvieron

información falsa sobre sus efectos.

Por otro lado modelo asigno un sentimiento negativo al tweet: *'Lo que tenés que saber sobre la vacunación contra el coronavirus <https://t.co/lnhwBpi7Ma> <https://t.co/xS7Idiyidy>'*, Esto es entendible debido a que este tweet a nuestro parecer es neutral ya que no reporta noticias negativas, ni positivas y esto se debe a que este usuario compartió información sobre la vacunación y esta no es posible analizarla.

De igual forma el modelo asigno un sentimiento negativo a el tweet *'Quién es la científica argentina que lidera desde el Reino Unido el desarrollo de fármacos contra el COVID-19 Por Valeria Román @ValeriaRoman Medio: Infobae <https://t.co/nGEoYhhMJN>'*, esta predicción es posible catalogarla como **falso negativo** por que se comparte información sobre los avances científicos en el desarrollo de medicinas para el COVID-19.

Como dato extra en nuestras pruebas de rendimiento se generó una validación cruzada de 5 iteraciones sobre el entrenamiento del modelo, esto con el objetivo de obtener una mejor percepción de su rendimiento en diferentes entrenamientos. La Tabla 4.1 muestra el promedio del desempeño alcanzado por la SVM en estas iteraciones.

	Precisión	Recall	F1-score
Promedio	0.74	0.70	0.71

Tabla 4.1: Promedio del desempeño de SVM

NOTA: Los valores de la Tabla 4.1 fueron calculados sin establecer el valor mínimo de precisión **"0.8"** indicado en la sección anterior.

En el siguiente y último capítulo abordaremos las conclusiones obtenidas en el desarrollo de esta tesis.

Capítulo 5

CONCLUSIONES

Durante la pandemia de COVID-19, la experiencia colectiva se caracterizó por un sentimiento generalizado de tristeza o estrés. Sin embargo, es necesario abstenerse de señalar a una única nación o región como responsable exclusiva de esta situación. A medida que se multiplicaban los comentarios en diversas plataformas y medios de comunicación, y ante un objetivo claro de mitigación, se reveló una falta de colaboración a nivel global para la búsqueda de soluciones colectivas. En lugar de trabajar en armonía para abordar la adversidad compartida, se observó una competencia por el desarrollo y la distribución de la primera vacuna aprobada por la Organización Mundial de la Salud (OMS), evidenciando un deseo de atribuirse el mérito de la ayuda a la comunidad global.

Desde la perspectiva de la tesis, el meollo del problema radicó en la deficiente comunicación interpersonal, un fenómeno arraigado en la naturaleza humana que se ve empeorado cuando se traslada al ámbito nacional e internacional. Esta deficiencia comunicativa genera tensiones políticas entre países, convirtiendo así la crisis sanitaria en un tema de mayor sensibilidad y complejidad.

Por otra parte, el procesamiento de lenguaje natural es herramienta para entender de manera resumida que quiere decir o transmitir una o cientos de personas cuando escriben u opinan sobre algo.

Con esta tecnología, en esta tesis se logró entrenar un modelo predictivo para poder diferenciar sentimientos en publicaciones durante la pandemia de COVID-19.

Para poder calibrar este modelo, se usaron datos de la plataforma Twitter, ahora llamada X, y se aplicó una metodología para análisis de sentimientos. El modelo entrenado es una SVM, cuyo desempeño promedio fue alrededor de 85% bajo la métrica de f1-score.

Se realizó una prueba más con datos no usados durante las fases de entrenamiento y pruebas, encontrándose una ligera degradación del desempeño del modelo, esto puede considerarse como un comportamiento esperado de los modelos predictivos en ambientes realistas.

Como parte del aprendizaje en el desarrollo de esta tesis, se encuentra la experiencia en la recolección de datos, lo que incluye la solicitud de datos en plataformas digitales y trámites para obtener cuenta como desarrollador; también la comprensión de conceptos y su implementación en lenguaje de programación Python; por último, la comprensión y aplicación de la metodología para análisis de sentimientos con datos reales en idioma español.

Referencias

- [1] AHMAD, K., AND IDE, N. *Affective Computing and Sentiment Analysis Emotion, Metaphor and Terminology*. Springer Science, Business Media Inc., 2011.
- [2] ALTINOK, D. *Mastering spaCy*. Packt Publishing, 2021.
- [3] ALZATE TAMAYO, A. *Programación estructurada un enfoque algorítmico*. Universidad Nacional de Colombia, 1997.
- [4] ARUMUGAM, R., AND SHANMUGAMANI, R. *Hands-On Natural Language Processing with Python : a Practical Guide to Applying Deep Learning Architectures to Your NLP Applications*. 2018.
- [5] BERZAL, F. *Estructuras de Control*, first ed. 2022.
- [6] CHAUCHAN, N. Métricas De Evaluación De Modelos En El Aprendizaje Automático, 2020.
- [7] CUEVA LOVELLE, J. M., GARCÍA FUENTE, M. D. P. A., LÓPEZ PÉREZ, B., LUENGO DIEZ, M. C., AND ALONSO REQUEJO, M. *Programación Estructurada y Orientada a Objetos con Pascal*, vol. 1. Borland España, 1994.
- [8] CUEVA LOVELLE, M. J., IZQUIERDO CASTANEDO, R., JUAN FUENTE, A. A., LUENGO DÍEZ, M. C., ORTÍN SOLER, F., AND LABRA GAYO, J. E. *Lenguajes, Gramáticas y Autómatas En Procesadores De Lenguaje*, first ed. SERVITEC, 2003.
- [9] DELGADO QUINTERO, S. Aprende Python. 1–488.

- [10] EXPLOSION.IA. Install spaCy, 2022.
- [11] FELDMAN, R., AND SANGER, J. *The Text Mining Handbook*. 2006.
- [12] FERRÉ GRAU, X., AND SÁNCHEZ SEGURA, M. I. Desarrollo Orientado a Objetos con UML. 1–53.
- [13] FOWLER, M., AND SCOTT, K. *UML Gota a Gota*. Pearson Education, Inc., 1999.
- [14] GANDH, R. Hiperplanos en el espacio de características 2d y 3d. https://miro.medium.com/v2/resize:fit:1100/format:webp/1*ZpkLQf2FNfzfH4HXeMw4MQ.png, 2018.
- [15] HAMILTON, K., AND MILES, R. *A Pragmatic Introduction to UML*, vol. 66. O’Reilly, 2006.
- [16] HERBRICH, R. *Learning Kernel Classifiers: Theory and Algorithms*. 2002.
- [17] HUNLOCK, P. Mastering JSON (JavaScript Object Notation). 1–11.
- [18] JACKSON, P., AND MOULINIER, I. Natural language processing for online applications: Text retrieval, extraction and categorization. *Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication* (2002), 237.
- [19] KAO, A., AND POTEET, S. R. *Natural language processing and text mining*. 2007.
- [20] KAVLAKOGLU, E. AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference?, 2020.
- [21] LARNER, A. *The 2x2 Matrix: Contingency, Confusion and the Metrics of Binary Classification*, vol. 1. 2021.
- [22] LIU, B. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press, 2015.

- [23] MARK, L. *Learning Python*, vol. 78. 2015.
- [24] MÁRQUEZ FRAUSTO, T. G., OSORIO ÁNGEL, S., AND OLVERA PÉREZ, E. N. *INTRODUCCIÓN A LA PORGRAMACIÓN ESTRUCTURADA EN C*, first edit ed. Pearson Educacion de México, Naucalpan de Juárez, Estado de México, 2011.
- [25] MARTELLI, A. *Python in a Nutshell*. O'Reilly, 2003.
- [26] POZZI, F. A., FERSINI, E., MESSINA, E., AND LIU, B. *Sentiment Analysis in Social Networks*. Todd Green, 2017.
- [27] PYTHON SOFTWARE FOUNDATION. Excepciones incorporadas, 2022.
- [28] PYTHON SOFTWARE FOUNDATION. string — Operaciones comunes de cadena de caracteres, 2022.
- [29] PYTHON SOFTWARE FOUNDATION. Tipos Integrados, 2022.
- [30] REAL ACADEMIA ESPAÑOLA. Diccionario de la lengua española, 2022.
- [31] RODRIGUEZ, A. ¿Qué es y para qué sirve JSON? Especificación oficial Javascript Object Notation Diferencia de XML.
- [32] RUMBAUGH, J., IVAR, J., AND BOOCH, G. *The Unified Modeling Language Reference Manual*, second ed., vol. 2. Pearson Education, Inc., Boston, MA, 2004.
- [33] RUSSELL, R. Machine Learning: Guia Paso a Paso Para Implementar Algoritmos de Machine Learning Con Python. 21–21.
- [34] SEVERANCE, C. R. *Python para todos*, vol. 1. Independently published, Ann Arbor, MI, USA, 2020.
- [35] STACK OVERFLOW CONTRIBUTORS. *Aprendizaje Regular Expressions*, first ed. No. Expresiones Regulares. RIP Tutorial, 2019.

- [36] TIOBE, T. S. Q. C. TIOBE Index for March 2022.
- [37] TWITTER INC. Developer Platform, 2022.
- [38] ULLAUDIOVISUAL - UNIVERSIDAD DE LA LAGUNA. Representación de documentos mediante TF-IDF, 2014.
- [39] W. BERRY, M., AND CASTELLANOS, M. *Survey of Text Mining II Clustering, Classification and Retrieval*. 2008.
- [40] WOLFEWICZ, A. Deep Learning vs. Machine Learning – What’s The Difference?, 2022.