Análisis de Sentimientos de Tweets en Español Basado en Técnicas Aprendizaje Supervisado

– –

Spanish Tweets Sentiment Analysis of based on Supervised Learning Techniques

TRABAJO FIN DE GRADO
CURSO 2020–2021

Juan Antonio Carrión García (Grado en Ingeniería Informática)
Rodrigo Fernández Ambrona (Grado en Ingeniería Informática)
Cristina Molina Gerbolés (Grado en Ingeniería Informática)
Patricia Motoso González (Grado en Ingeniería de Computadores)

Directores

Francisco Javier Crespo Yáñez
Luis Javier García Villalba

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid
Madrid, Junio de 2021

Agradecimientos

A nuestros directores de proyecto, Francisco Javier y Luis Javier que mostraron, desde el primer día que comenzamos el proyecto, una gran confianza en lo que podríamos conseguir; y animándonos semanalmente pero ir avanzado poco a poco.

A Esteban por todo el apoyo técnico proporcionado durante todo el proyecto.

A Ana L. Sandoval por su ayuda con todo el tema de documentación, además de su apoyo constante.

A diversos profesores que han estado durante todos estos años de carrera y con los que hemos podido adquirir conocimientos necesarios para poder desarrollar este trabajo.

A la Facultad de Informática de la Universidad Complutense de Madrid por habernos ofrecido herramientas online en momentos de COVID19 con los que hemos podido avanzar en el trabajo sin tener que acudir presencialmente.

Índice General

Índice de Figuras IX

Índice de Tablas XI

Lista de Acrónimos XIII

Abstract XV

Resumen XVII

1. Introducción 1

1.1. Motivación . . . 1
1.2. Contexto . . . 2
1.3. Objetivos . . . 2
1.4. Plan de Trabajo . . . 3
1.5. Estructura del Trabajo . . . 4

2. Técnicas de Análisis de Datos 7

2.1. Evolución de la Ciencia de los Datos . . . 7
2.1.1. Big Data en el Contexto Actual . . . 8
2.2. Técnicas de Aprendizaje Automático . . . 9
2.2.1. Técnicas de Aprendizaje Supervisado . . . 12
2.2.2. Técnicas de Aprendizaje no Supervisado . . . 20
2.2.3. Aprendizaje Profundo . . . 21
2.3. Métricas Utilizadas en Problemas de Clasificación . . . 23
2.4. El problema del Sobreaprendizaje . . . 25
2.4.1. Cómo Prevenir el Sobreaprendizaje . . . 26

3. PLN y Análisis de sentimientos 29

3.1. Definición de Lenguaje Natural . . . 29
3.1.1. Niveles de Analisis en el Procesamiento del Lenguaje Natural . . . 30
3.2. Procesamiento Computacional del Lenguaje Natural . . . 31
3.2.1. Conceptos del Procesamiento del Lenguaje Natural . . . 31

3.2.2. Aplicaciones del Procesamiento del Lenguaje Natural 32
3.2.3. Definición de Análisis de Sentimientos 33

4. Estado del Arte **39**
4.1. Técnicas Basadas en Algoritmos que Utilizan Distintas Herramientas 39
4.2. Técnicas Basadas en el Léxico 40
4.3. Técnicas Basadas en la Combinación del Léxico y el Aprendizaje Automático 41
4.4. Técnicas Basadas en el Aprendizaje Automático 42
4.5. Técnicas basadas en el Uso de Dataset en Castellano en el Análisis de Sentimientos en Twitter 44

5. Metodología de Análisis de Sentimientos Propuesta **47**
5.1. Preprocesado del Corpus 47
5.1.1. Eliminación de las Stop Words 48
5.1.2. Correción Ortográfica 48
5.1.3. Etiquetado 49
5.2. Descripción del Modelo 52
5.2.1. Paso de los Datos a Frecuencias TF-IDF 52
5.2.2. Entrenamiento y Comparación de Clasificadores 52
5.2.3. Predicción 53
5.3. Parametrización de los Algoritmos 54
5.3.1. Descenso de Gradiente Estocástico 55
5.3.2. Aumento de Gradiente Extremo 55
5.3.3. Los K Vecinos más Cercanos 55
5.3.4. Árboles de Clasificación 56
5.3.5. Bosques Aleatorios 56
5.3.6. Clasificador Bayesiano Ingenuo Multinomial 56
5.3.7. Máquina de Soporte Vectorial 57
5.3.8. Clasificador Bayesiano Ingenuo de Bernoulli 57
5.3.9. Regresión Logística 57
5.4. Experimentos 58
5.4.1. Evaluación del Algoritmo Descenso de Gradiente Estocástico 58
5.4.2. Evaluación del Algoritmo Aumento del Gradiente Extremo 59
5.4.3. Evaluación del Algoritmo K Vecinos más Cercanos 60
5.4.4. Evaluación del Algoritmos de Árboles de Decisión 61
5.4.5. Evaluación del Algoritmo Bosques Aleatorios 62
5.4.6. Evaluación del Algoritmo Bayesiano Ingenuo Multinomial 63
5.4.7. Evaluación del Algoritmo Regresión Logística 64
5.4.8. Evaluación del Algoritmo Clasificador de Vector de Soporte 65

5.4.9. Evaluación del Algoritmo Bayes Ingenuo de Modelos de Bernoulli Multivariados . . . 66
5.4.10. Comparación de Resultados y Análisis de Tiempos . . . 66

6. Contribuciones Individuales **71**
6.1. Rodrigo Fernández Ambrona . . . 71
6.2. Cristina Molina Gerbolés . . . 73
6.3. Patricia Motoso González . . . 74
6.4. Juan Antonio Carrión García . . . 75

7. Conclusiones y Trabajo Futuro **79**
7.1. Conclusiones . . . 79
7.2. Trabajos futuros . . . 80

8. Introduction **83**
8.1. Introduction and context . . . 83
8.2. Context . . . 84
8.3. Goals . . . 84
8.4. Workplan . . . 85
8.5. Work Structure . . . 86

9. Conclusion **87**
9.1. Conclusions . . . 87
9.2. Future work . . . 88

Bibliografía **89**

Índice de Figuras

1.1. Gráfica de la evolución de los usuarios de Twitter 2
1.2. Diagrama de Gantt 4

2.1. Diagrama de unión del aprendizaje automático y la ciencia de datos 10
2.2. Representación de un árbol de clasificación C4.5 12
2.3. Representación de Random Forest 14
2.4. hyperplanos que dividen los dos grupos de puntos 15
2.5. Comportamiento de las diferentes versiones de Naive Bayes 16
2.6. Representación de una red neuronal 23
2.7. Gráficas que muestran el subaprendizaje, sobreaprendizaje y un modelo bien entrenado 25

3.1. Esquema de comunicación (Cortez Vasquez et al. 2009) 30
3.2. Niveles de Análisis (Cortez Vasquez et al. 2009) 31
3.3. Metodología general de análisis de sentimiento. 35

5.1. Diagrama de la metodología 47
5.2. Gráficas usando TF/IDF 52
5.3. Diagrama del modelo 54
5.4. Gráfica de la media de los tiempos por clasificador usando conjunto de *train* 67

8.1. Graphic of the evolution of Twitter users 84

Índice de Tablas

2.1. Parámetros usados en el árbol de clasificación . . . 13
2.2. Parámetros usados en el algoritmo de bosques aleatorios . . . 14
2.3. Parámetros usados en el clasificador bayesiano ingenuo . . . 16
2.4. Parámetros usados en el algoritmo de regresión lineal . . . 17
2.5. Parámetros usados en el algoritmo de regresión logística . . . 18
2.6. Parámetros usados en el algoritmo XGBoost . . . 18
2.7. Parámetros usados en el algoritmo K vecinos más cercanos . . . 19
2.8. Parámetros usados en el algoritmo K medias más cercanos . . . 21
2.9. Matriz de confusión . . . 25

3.1. Herramientas y librerías para PLN . . . 35

4.1. Tabla comparativa . . . 46

5.1. VADER con traductor de Google . . . 51
5.2. VADER con traductor de Bing . . . 51
5.3. Textblob con traductor de Google . . . 51
5.4. Textblob con traductor de Bing . . . 51
5.5. Comparativa resultados Vader y TexBlob . . . 51
5.6. Mejores 5 configuraciones del modelo con el algoritmo SGD . . . 58
5.7. Matriz de confusión con los resultados del test con el algoritmo SGD . . . 59
5.8. Resultados de las métricas para el *test* con el algoritmo SGD . . . 59
5.9. Mejores configuraciones del modelo con el algoritmo Aumento del Gradiente Extremo . . . 59
5.10. Matriz de confusión del test con el algoritmo Aumento del Gradiente Extremo 60
5.11. Resultados de las métricas para test con el algoritmo Aumento del Gradiente Extremo . . . 60
5.12. Matriz de confusión con los resultados del test con el algoritmo K Vecinos más Cercanos . . . 60
5.13. Resultados de las métricas para el test con el algoritmo K Vecinos más Cercanos . . . 61
5.14. Mejores 5 configuraciones del modelo con el algoritmo K Vecinos más Cercanos 61

5.15. Mejores 5 configuraciones del modelo con el algoritmo árboles de decisión . 61
5.16. Matriz de confusión del test con el algoritmo árboles de decisión 62
5.17. Resultados de las métricas para test con el algoritmo árboles de decisión . . 62
5.18. Mejores 5 configuraciones del modelo con el algoritmo Bosques Aleatorios . 62
5.19. Matriz de confusión del test con el algoritmo Bosques Aleatorios 63
5.20. Resultados de las métricas del test con el algoritmo Bosques Aleatorios . . . 63
5.21. Configuraciones del modelo con el algoritmo Multinomial 63
5.22. Matriz de confusión con los resultados del test con el algoritmo Multinomial 64
5.23. Resultados de las métricas con el algoritmo Multinomial 64
5.24. Configuraciones del modelo con el algoritmo Regresión Logística 64
5.25. Matriz de confusión del test con el algoritmo Regresión Logística 64
5.26. Resultados de las métricas para el test con el algoritmo Regresión Logística 65
5.27. Configuraciones del modelo con el algoritmo SVC 65
5.28. Matriz de confusión del test con el algoritmo SVC 65
5.29. Resultados de las métricas del test con el algoritmo SVC 66
5.30. Configuraciones del modelo con el algoritmo Bernoulli Naive Bayes 66
5.31. Matriz de confusión del test con el algoritmo Bernoulli Naive Bayes 66
5.32. Resultados de las métricas del test con el algoritmo Bernoulli Naive Bayes . 67
5.33. Resultados de los diferentes algoritmos propuestos 67
5.34. Tabla comparativa 68
5.35. Tabla comparativa trabajos más relacionados 69

Lista de Acrónimos

AS	Análisis de Sentimientos
FN	Falso Negativo
FP	Falso Positivo
IA	Inteligencia Artificial
IDF	Inverse Document Frecuency
KNN	K Nearest Neighbors
LF	Lenguaje Formal
LLR	Logistic Linear Regression
LN	Lenguaje Natural
NB	Naive Bayes
NLTK	Natural Language Processing Toolkit
PLN	Procesamiento del Lenguaje Natural
ReLU	Rectified Linear Unit

RF	Random Forest
SGD	Stochastic Gradient Descent
SVC	Support Vector Classifier
SVM	Support Vector Machine
TF	Term Frecuency
VADER	Valence Aware Dictionary y Sentiment Reasoner
VN	Verdadero Negativo
VP	Verdadero Positivo
XGBoost	Extreme Gradient Boosting

Abstract

In the last few years, platforms like Twitter or Facebook have become more important and people using them has increased exponentially. In 2021 was sent an average of 500 million twetts every day Twitter (2020). In this context, with so much information and proccesing increased, many companies have started minning social media due to the many advantages it has. This has caused an increase on the importance of Sentiment Analysis when you want to know what people thinks about a topic. Using a dataset of spanish tweets with two labels(positive and negative), in this paperwork we are going to classify another label to extract more information from the data. To add this new label an authomatic labeling was used. Due to the fact that Valence Aware Dictionary y Sentiment Reasoner (VADER) and TextBlob labeling works only on English text, a previous translation was made using Google and Bing translators. In other words, four different combinations was tested and only the best one was chosen to add the new label. The next step was to find out the best number of words using Term Frecuency (TF)-Inverse Document Frecuency (IDF) matrix, consequently transforming the corpus using that matrix. Finally, several classifiers were tested to get the best one predicting the correct tweet labels. The results obtained were the following ones: The best translator-labeler combination was Bing and VADER respectively, labeling 39.4 % of the original dataset as neutral and having 67 % accuracy when excluding neutral tweets. The best classifier was Stochastic Gradient Descent with 75.94 % accuray, 76.53 % precision on average and 74.5 % recall.

Keywords: Sentiment Analysis, Category, Classifier, TF-IDF Matrix, Translator.

Resumen

Durante los últimos años, plataformas tales como *Twitter* o *Facebook*, han ganado fuerza y el número de personas que las usan ha crecido exponencialmente. En el año 2021 se envían de media alrededor de 500 millones de *Tweets* al día Twitter (2020). En este contexto, con tal cantidad de información, y gracias al aumento de la capacidad de procesamiento, muchas empresas, atraídas por las ventajas que conllevaba, han optado por minar información de estas redes sociales. Esto hace que el campo del Análisis de Sentimientos cobre gran importancia a la hora de obtener lo que la gente piensa acerca de algo. Este trabajo consiste en, dado un *dataset* de *Tweets* en castellano etiquetado con dos categorías (positiva y negativa) según el sentimiento del tweet, añadir una categoría más para ampliar la información que pudiera extraerse de los datos. En este paso de añadir la nueva categoría, se utiliza un etiquetador automático. Dado que los etiquetadores VADER y *Textblob* funcionan solo con textos en inglés, se hace una traducción con los traductores de *Google* y de *Bing*. Es decir, se prueban las cuatro combinaciones, y la mejor de estas es la que se usa finalmente para etiquetar esta nueva categoría. A continuación, se comprueba cual es el mejor número de palabras a la hora de crear la matriz TF-IDF, y a continuación se transforma el corpus a dicha matriz. Finalmente, se analiza una serie de clasificadores para obtener uno que sea capaz de predecir lo mejor posible la categoría a la que pertenece un *tweet*. Los resultados fueron los siguientes: la mejor combinación traductor-etiquetador fue el traductor de *bing* y el etiquetador VADER, obteniendo solo un **39,4 %** de *tweets* etiquetados como neutros, y un **67 %** de acierto con el *dataset* original. El mejor clasificador fue el Descenso del Gradiente Estocástico, el cual obtuvo una *accuracy* del **75,94 %**, una *precision* media del **76,53 %** y un *recall* del **74,5 %**.

Palabras clave: Análisis de Sentimientos, Categoría, Clasificador, Matriz TF-IDF, Traductor

Capítulo 1

Introducción

1.1. Motivación

A día de hoy, existen más de 353 millones de usuarios activos en la red social *Twitter*. En una red social basada en la publicación de opiniones acerca de temas de importancia actual, la capacidad de tener una vista general acerca de la opinión pública se vuelve extremadamente útil.

Durante los últimos años, ha habido un exponencial crecimiento del número de personas que le daban uso a este tipo de plataformas, como muestra la Figura 1.1, dedicadas a enviar estas opiniones en mensajes de una longitud limitada. Impresionadas por dicho crecimiento, las grandes compañías se interesaron y procedieron a buscar formas de minar y extraer datos de Twitter para poder sacar información acerca de lo que la gente piensa.

En este contexto se le puede dar uso a dos campos de la inteligencia artificial: El Análisis de Sentimiento (AS) y el Procesamiento del Lenguaje Natural (PLN). Algunas de las aplicaciones que se le pueden dar podrían ser realizar investigaciones y análisis de mercado, o simplemente saber la opinión de los clientes acerca de un cierto producto o servicio. Todo esto, dando unos resultados muy buenos y cercanos a la realidad, debido a la gran cantidad de datos que se pueden procesar en tiempos razonables a día de hoy.

Por otra parte, una de las ventajas de estos dos campos de estudio del Aprendizaje Automático es que son extrapolables a problemas muy diversos. Tanto es así, que un clasificador entrenado con técnicas de Análisis de Sentimiento y Procesamiento del Lenguaje Natural, introduciéndole un corpus suficientemente grande y variado de *Tweets*, puede ser usado en otros campos, tales como en la tarea de la predicción de sentimiento en un *dataset* de llamadas.

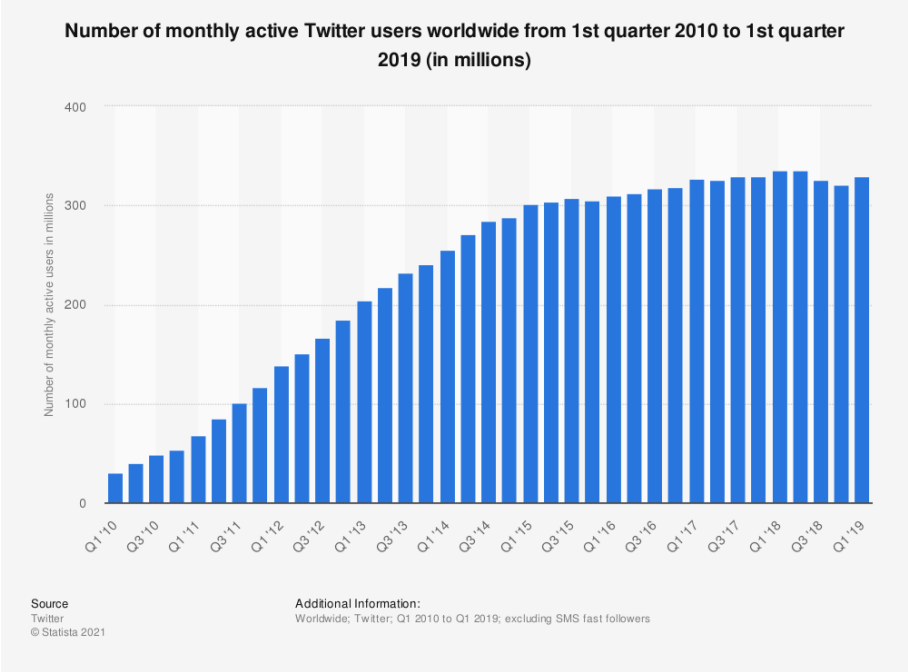


Figura 1.1: Gráfica de la evolución de los usuarios de Twitter

1.2. Contexto

Este Trabajo Fin de Grado ha sido realizado dentro del Grupo de Análisis, Seguridad y Sistemas (Grupo GASS, `https://gass.ucm.es/`, Grupo 910,623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación THEIA (Techniques for Integrity and Authentication of Multimedia Files of Mobile Devices) con referencia FEI-EU-19-04.

1.3. Objetivos

El objetivo de este proyecto es el diseño e implementación de un algoritmo de análisis de sentimientos basado en textos, en el que se tomará como referencia el trabajo escrito en castellano de (Holgado et al. 2020). Se va a mejorar los resultados de dicho *paper* de investigación, añadiendo una categoría más para poder obtener mayor información de los datos, y a su vez buscar un algoritmo que clasifique *tweets* en español de la mejor forma posible.

El siguiente objetivo a considerar es cambiar el tipo de preprocesado para comprobar si los resultados mejoran, a su vez se toma como finalidad hacer un etiquetado automático usando dos herramientas para ello, *VADER* y *TextBlob* y compararlas para examinar cual da mejor rendimiento.

Se considera vocabulario al conjunto de las n palabras con mayor relevancia a la hora de clasificar un *tweet*. Otro objetivo de este trabajo, es obtener el número óptimo de

palabras que debe tener dicho vocabulario para luego entrenar los clasificadores. Además se propone aumentar el número de clasificadores usados en el trabajo anteriormente menciona, aumentando el número de clasificadores para comparar, así como añadir algunas técnicas que están más a la vanguardia del aprendizaje automático como pueden ser las redes neuronales.

1.4. Plan de Trabajo

En esta sección se explican las fases que han tenido lugar durante todo el proyecto.

1. **Investigación**: La primera fase de estado del arte se llevó a cabo durante un periodo de aproximadamente 4 meses, se buscaron papers, documentos y trabajos relacionados con la temática del TFG, en este caso sobre Procesamiento del Lenguaje Natural y sobre Análisis de Tweets. Por otra parte, si bien es cierto que también se investigaron algunos papers de temas de tratamiento del sonido y paso de audio a texto, finalmente esa parte del trabajo fue delegada a otro grupo, encargado del TFG "Analisis de llamadas II". A partir de entonces, el trabajo se centró en la parte de Procesamiento del Lenguaje Natural. Cabe destacar que en esta fase se hizo la búsqueda de todos los documentos explicados posteriormente en el capítulo 5. Por último, en esta fase se estudió el lenguaje Python, además de la descarga de todas las librerías y entornos que más adelante fueron necesarios a la hora de implementar el código para realizar los experimentos necesarios *Scikit-learn, PyCharm,...*

2. **Desarrollo**: En esta segunda fase se comenzó la implementación de lo que terminaría siendo el código final. A partir de un *dataset* de *tweets*, se procedió a preprocesarlo hasta llegar a tener un conjunto de datos que los clasificadores fueran capaces de usar a la hora de ser entrenados. Inicialmente los tweets solo estaban etiquetados en las categoría positiva y negativa. Se añadió una nueva categoría, la neutra, a las dos que inicialmente había, etiquetando algunos de los tweets como tweets neutros. Por otro lado, se hizo un tratamiento del *dataset* para que más adelante los clasificadores no tuvieran que lidiar con problemas intrínsecos de los datos, tales como la eliminación de carácteres especiales, de menciones y de hashtags, y de palabras que no aportaban (las *stopwords*), además de normalizar el texto. Una vez preprocesados los datos, se diseñó el modelo para facilitar el test de clasificadores.

3. **Experimentos**: En la fase de experimentos lo primero que se hizo fue comprobar cual el número óptimo de palabras para separar el texto en categorías. Tras eso, los datos del corpus se transformaron a frecuencias numéricas. Y ya por último, con dichas frecuencias, se procedió a probar todas las combinaciones representativas de los hiperparámetros de los clasificadores, con la intención de tener un amplio

abanico de resultados y poder escoger con criterio cuales eran los mejores. A dichos clasificadores se los entrenó con la muestra de *train*.

4. **Resultados**: Finalmente, se extrajeron resultados para ver qué clasificadores eran los mejores. Se comprobó que dichos clasificadores daban un rendimiento similar con la muestra de *test*, y que por ende, no habían sido sobreentrenados. A partir de ahí, se extrajeron conclusiones, comparando nuestros resultados con los de trabajos similares.

La Figura 1.2 muestra el diagrama de Gantt correspondiente al flujo de trabajo que se ha ido realizando.

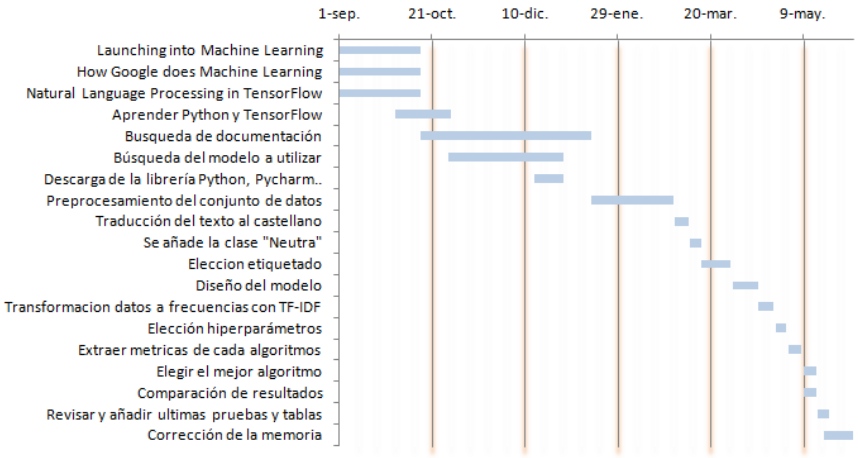


Figura 1.2: Diagrama de Gantt

1.5. Estructura del Trabajo

La memoria de aquí en adelante está organizada en 5 capítulos.

El capítulo 2 explica conceptos introductorios a la ciencia de datos así como conceptos básicos del aprendizaje automático. En este capítulo también se tratan los diferentes clasificadores que se van a usar durante todo el proyecto. Por último se analizan las métricas utilizadas para medir la calidad de una clasificación y el problema que existe con el sobreaprendizaje y como prevenirlo.

El capítulo 3 describe el procesamiento del lenguaje natural y el análisis de sentimientos. Se definen los diferentes conceptos de Procesamiento del Lenguaje Natural (PLN) que aparecerán durante todo el proyecto, algunas aplicaciones del PLN y del Análisis de Sentimientos (AS).

El capítulo 4 analiza el estado actual de la investigación en PLN y AS comparando las diferentes técnicas utilizadas con diversos trabajos.

El capítulo 5 explica la construcción del modelo utilizado, así como todos los pasos previos que se han tenido que realizar para poder llegar a tener un corpus susceptible a clasificar. También se explican los parámetros utilizados de los algoritmos que se han seleccionado para la tarea de clasificar y por último se detallan los experimentos realizados junto a las conclusiones que se sacan de estos.

El capítulo 7 describe las conclusiones de todo el proyecto de investigación realizado, así como la propuesta de trabajos futuros.

Finalmente, los capítulos 8 y 9 presentan un resumen en inglés de la introducción y las conclusiones.

Capítulo 2

Técnicas de Análisis de Datos

Para poner en práctica el análisis de llamadas primero hay que comprender los aspectos teóricos sobre las técnicas de análisis de datos relevantes para el desarrollo de este trabajo. En la Sección 2.1 se muestra la evolución del área conocida como ciencia de los datos. Seguidamente, en la Sección 2.2 se presenta una breve introducción al aprendizaje automático, una de las dos áreas de Inteligencia artificial tratadas en este trabajo, junto con las cuatro técnicas que conforman el área de aprendizaje automático. En la Sección 2.3 se detallan las métricas que se van a utilizar en el los problemas de clasificación. Finalmente en la Seccion 2.4 se muestra cual va a ser el problema que puede tener la utilización de un aprendizaje supervisado.

2.1. Evolución de la Ciencia de los Datos

Desde los inicios de la computación, se ha considerado la informática como la herramienta para automatizar problemas que eran demasiado grandes como para que una persona humana se encargara de ellos. Los ordenadores trabajan con números y, en principio, salvo que nosotros se lo programemos específicamente, carecen de posibilidad de abstracción y de entender un problema cualquiera que se le pueda plantear. Con lo cual es tarea del informático extraer una aproximación matemática del problema e introducírsela al ordenador a través de un algoritmo plasmado en líneas de código. Estas líneas de código serán traducidas por un intérprete o compilador a instrucciones de bajo nivel. Finalmente el ordenador ejecutará esas instrucciones o lo que es lo mismo, el programa.

Existe una serie de problemas que tienen como finalidad extraer patrones de una colección de sucesos de la vida real. Estos problemas normalmente eran abordados desde la estadística, pero gracias a la herramienta de la computación, todos los cálculos que antes se tenían que hacer a mano ahora están a cargo de una máquina capaz de ejecutarlos automáticamente. Todo este conjunto de técnicas que mezclan conceptos de estadística, informática, matemáticas y ciencias de la información, constituyen lo que llamamos Ciencia

de Datos.

El concepto Ciencia de Datos fue utilizado por primera vez en la década de los años 60 por el científico danés Peter Naur. Aunque se pueda pensar que Ciencia de Datos es algo completamente nuevo, la realidad es que es un área con unos 60 años de investigación a las espaldas. Sin embargo, durante la última década ha sido cuando ha desarrollado todo su potencial debido a ciertos factores.

La primera razón del estancamiento de la ciencia de datos durante años fue el poco interés que suscitaba fuera de la investigación. Esto se debía a que a la hora de aplicar tecnologías, tales como el perceptrón simple, a casos de la vida real, se topaba con problemas que en aquel entonces no tenían solución.

Frank Rosenblatt creó el perceptrón simple en 1958 (Biehl & Riegler 1994), un algoritmo de reconocimiento de patrones cuyo propósito es clasificar un conjunto de elementos X de los cuales conocemos que tienen los campos $x_1, x_2, x_3, ..., x_n$ en dos clases (y_1 e y_2). Este algoritmo se basa en la creación un hyperplano, el cual se utilizará como superficie de decisión en función de si un punto dentro de un espacio vectorial de n dimensiones, con n el número de variables que hayamos utilizado, se encuentra a un lado u otro del hyperplano. Las limitaciones de este algoritmo vinieron cuando se le plantearon problemas que requerían mayor complejidad que un hyperplano, como es el caso del Reconocimiento de la XOR (Singh 2016). Este problema frenó la evolución de la investigación en este campo. Más adelante sería resuelto con el Algoritmo de retropropagación (Hecht-Nielsen 1992).

Otra razón por la que este campo no suscitó interés hasta hace relativamente poco, es el hecho de que para hacer un predictor que tenga una visión cercana a la realidad y que resuelva el problema que se le plantea, necesita una muestra significativa de los datos. Es decir, datos que se asemejen lo máximo posible a la realidad. La forma de que un algoritmo aprenda y extraiga conocimiento lo más cercano posible a la realidad es introducir una muestra suficientemente grande de datos. Además, otra ventaja que aporta el hecho de que la muestra de datos sea voluminosa, es que contribuye a evitar que el modelo o algoritmo que entrenemos sobreaprenda los datos. Hasta hace poco, no existían tecnologías para mitigar los problemas de coste en tiempo y espacio que conlleva el procesamiento de datos en cantidades masivas.

2.1.1. Big Data en el Contexto Actual

Se define *Big Data* como un conjunto de datos, o combinaciones de datos, cuyo tamaño, complejidad y velocidad de crecimiento impiden su procesamiento usando tecnologías y herramientas convencionales. Este concepto cobró importancia a principios de la década de 2000 (Powerdata 2020).

La razón de la gran importancia de *Big Data* en el mundo empresarial, es el hecho de que al tener tantos datos, la información que se extraerá de ellos será más precisa y correcta.

Con ello se podrán resolver mejor los problemas que requieran de dicha información. Con esto pueden identificar aquellas áreas más problemáticas antes de que interfieran en su reputación o beneficios. Las empresas que tienen más éxito a la hora de emplear tecnologías *Big Data* obtienen los siguientes beneficios (Powerdata 2020):

- **Nuevos servicios y productos:** Con el Big data muchas empresas estan creando nuevos productos y servicios que satisfagan las necesidades de los usuarios ya que es posible medir la satisfacción de los clientes.
- **Reducir el coste:** Las grandes tecnologías de datos, como el análisis basado en la nube, tienen muchas ventajas a la hora de almacenar grandes cantidades de datos ya que reducen costes de almacenamiento.
- **Rapidez:** Las empresas pueden analizar la información de forma inmediata.

2.1.1.1. Desafíos de la Calidad de los Datos

Es importante tener datos de la mejor calidad posible, dado que de ello depende en gran medida si se tendrá que hacer un preprocesamiento para tratarlos o no, con todo lo que ello conlleva.

A continuación se van a enumerar una serie de desafíos a los que se enfrentan la calidad de datos de Big Data (Powerdata 2020):

- **Gran número de fuentes y tipos de datos:** Se tienen fuentes de datos muy amplias como por ejemplo: datos de internet, datos de móvil, datos experimentales. Por otro lado los tipos de datos suelen ser en su mayoría NO ESTRUCTURADOS (documentos, videos, audios...) y esto produce un mayor número de errores si no ideamos un proyecto de calidad en los datos.
- **Muy volatil:** Los datos cambian rápidamente por lo que tienen una validez muy corta. Es necesario solucionarlo usando un procesamiento muy alto.
- **Gran volumen de datos:** El volumen de datos es muy grande y esto dificulta la ejecución de un proceso dentro de un tiempo razonable.
- **No hay estándares de calidad de datos que estén unificados:** Estas normas necesitan actualizarse ya que no fue hasta el 2011 cuando ISO publicó las normas de calidad de datos.

2.2. Técnicas de Aprendizaje Automático

El aprendizaje automático es una aplicación de la inteligencia artificial. Se considera que las máquinas aprenden y pueden predecir resultados en base a experiencias pasadas (Caparrini 2017). Los investigadores buscan generar algoritmos que generalicen

comportamientos y reconozcan patrones a partir de una información proporcionada como entrada de estos algoritmos. El aprendizaje automático y la ciencia de datos están profundamente relacionadas ya que la ciencia de datos es un campo muy grande que abarca múltiples disciplinas, una de ellas es el aprendizaje supervisado (de Ciencia de Datos 2020). Tal como se muestra en la Figura 2.1 esto se puede solapar con la minería de datos ya que el fin de ambas disciplinas es el análisis de datos pero el aprendizaje automático se centra más en aspectos prácticos y no puramente teóricos.

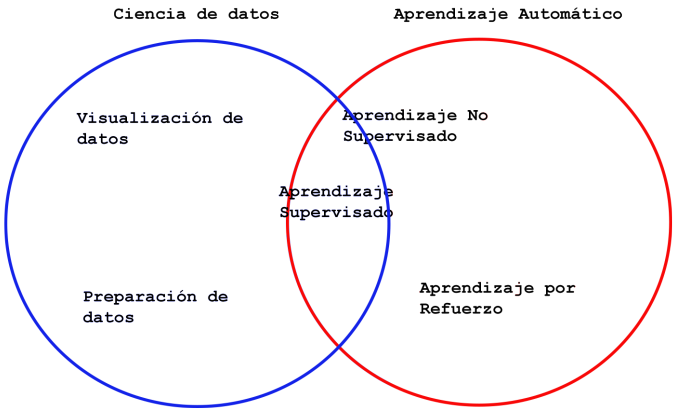


Figura 2.1: Diagrama de unión del aprendizaje automático y la ciencia de datos

Una de las formas populares en la actualidad para examinar los comportamientos emocionales en los textos es haciendo uso del aprendizaje automático o por su terminología en el inglés *machine learning*. Se trata de un subcampo de la ciencia de la computación y la inteligencia artificial cuyo funcionamiento se centra en la detección de patrones en los datos para la creación de algoritmos inteligentes que puedan aprender sin depender de la programación basada en reglas, estos algoritmos tienen la capacidad de mejorar con el paso del tiempo de una forma autónoma, y se pueden dividir en diferentes modelos (Li 2021):

- **Modelo geométrico**: Construidos en el espacio, pueden tener una, dos o más dimensiones, si hay una línea de separación entre las clases decimos que los datos son linealmente separables.
- **Modelo probabilístico**: Se usa las prioridades para poder clasificar los datos, la probabilidad Bayesiana es uno de los pilares fundamentales de este modelo.
- **Modelo lógico**: Expresan las probabilidades de forma lógica organizados en árboles de decisión.

También se pueden distinguir diferentes técnicas para realizar esta tarea, aquí presentamos alguno de ellos (de los Santos 2017):

- **Aprendizaje supervisado**: Los algoritmos trabajan con conjuntos de datos etiquetados, buscando una función que, dados los datos de entrada, predigan el etiquetado adecuado como salida. Para este tipo de aprendizaje automático se necesita intervención humana ya que los datos de entrada están previamente etiquetados. En respuesta a esto el algoritmo genera datos de salida esperados basándose en este subconjunto de datos pasados, de esta manera el modelo aprende a clasificar. Dos tipos de datos se pueden introducir a estos algoritmos (Calvo 2019):

  - Datos de clasificación que clasifican un objeto en diferentes clases.
  - Datos de regresión que predicen un valor numérico.

  Algunos algoritmos de aprendizaje automático son: algoritmos genéticos, aumento del gradiente extremo, Bayes ingenuo... Este tipo de aprendizaje automático es el que se va a usar en este trabajo (Bismart 2021).

- **Aprendizaje no supervisado**: Encuentra similitudes o características en común en el conjunto de datos y estas definen el etiquetado. Al contrario que para el aprendizaje supervisado, para el no supervisado no contamos con un *dataset* etiquetado para poder entrenar nuestra máquina. No cuenta para nada con intervención humana (Bismart 2021). Para buscar las características comunes se tiene en cuenta una métrica, por ejemplo la distancia entre los individuos. Una de las ventajas de utilizar este tipo de aprendizaje, es que los conjuntos de datos para entrenamiento son menos costosos de conseguir. Los dos tipos de algoritmos que utilizan el aprendizaje no supervisado son (Roman 2019):

  - *Clustering*: Clasifica los datos de salida en grupos.
  - *Asociación*: Descubre reglas y patrones dentro del conjunto de datos.

- **Aprendizaje por refuerzo**: Se inspira en la corriente psicológica del conductismo. Se le ofrece a la máquina premios o castigos según las decisiones que tome, siendo la finalidad de la máquina conseguir la mayor cantidad de premios. Este tipo de algoritmo no se puede considerar supervisado ya que no tenemos datos etiquetados pero tampoco podemos clasificarlo como algoritmo no supervisado por que no buscamos características en común para agrupar individuos (Bagnato n.d.).

- **Aprendizaje profundo**: El aprendizaje profundo presenta un sistema mucho más potente que el resto de tipos de algoritmos ya que no posee una capacidad finita para aprender independientemente de la cantidad de datos que reciban (NetApp 2021).

2.2.1. Técnicas de Aprendizaje Supervisado

Las técnicas de aprendizaje supervisado que se han utilizado en este trabajo son las siguientes: Árboles de clasificación, Árboles Aleatorios, Máquinas de soporte vectorial, Clasificador Bayesiano Ingenuo, Regresión Lineal y Logística y Aumento de Gradiente Extremo.

2.2.1.1. Árbol de clasificación

Dentro de que hay muchos tipos de árboles de clasificación, nos centraremos en el que se construye mediante el algoritmo ID3. El algoritmo ID3 construye el árbol de decisión de manera descendente. A cada paso que da, consulta todos los atributos, viendo vorazmente cuál discrimina mejor, bifurcando el conjunto de datos en dos subconjuntos. Las métricas que se suelen usar para saber cómo de bien discrimina un atributo son el Coeficiente de Gini, referenciado en la ecuación 2.1 y la Entropía en ecuación 2.2.

Supongamos p_j la probabilidad de cada clase.

$$Gini = 1 - \sum_j p_j^2 \tag{2.1}$$

$$Entropia = -\sum_j p_j * log_2(p_j) \tag{2.2}$$

Existe una extensión del algoritmo ID3 llamada C4.5 (Figura 2.2). Este algoritmo, al contrario que el ID3, está basado en *condiciones booleanas*. Es decir, dentro de cada nodo del arbol, recorre todas las muestras para ver cual es la que mejor separa.

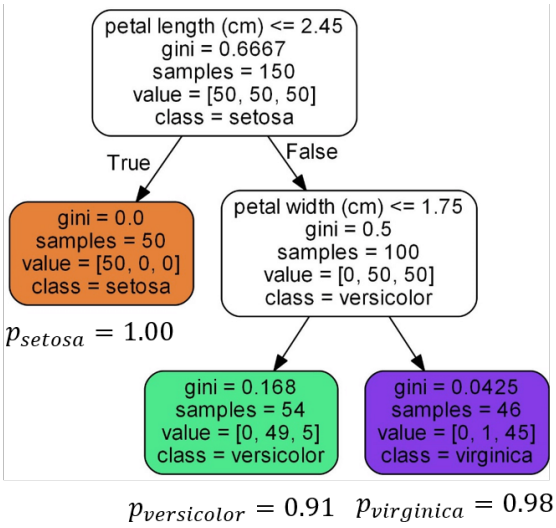


Figura 2.2: Representación de un árbol de clasificación C4.5

Al igual que en K Nearest Neighbors (KNN), en cada conjunto o subconjunto que se forme, se tomará la clase como aquella que tenga una mayoría de elementos pertenecientes

a ella. Esto lo hace hasta que solo quedan atributos de una única clase en cada nodo del árbol. Otra opción es limitarlo ajustando parámetros como número de nodos hoja del árbol, o mínimo tamaño de los datos dentro de cada hoja.

Este tipo de parámetros se suelen ajustar adrede para evitar que el árbol sobreaprenda los datos. (Pedregosa et al. 2011) Los parámetros que *Sklearn* nos permite pasarle a los árboles de decisión se muestran en la Tabla 2.1.

Tabla 2.1: Parámetros usados en el árbol de clasificación

Parámetro	**Descripción**
Criterion	Es la función que mide cómo de bueno es la división que ha hecho el árbol en cada nodo. Puede tomar los valores 'gini' y 'entropy', por defecto usará gini.
Splitter	Se refiere a la estrategia usada para elegir como se van a particionar los datos. Puede tomar 'random' o 'best' como valores, por defecto tiene 'best'. Con 'best' se elige la mejor partición y con 'random' se elige una partición de forma aleatoria.
Max_depth	Es la profundidad máxima que podrá tomar el árbol, cuando su valor es *None*, el árbol podrá seguir bajando hasta que encuentre nodos puros o las hojas contengan menos del número mínimo de muestras permitido. Es uno de los parámetros que nos ayuda a controlar el sobreaprendizaje, ya que si conseguimos nodos puros no será capaz de generalizar. Por defecto toma de valor *None*
Min_samples_split	Como valor por defecto toma 2. Puede tomar valores de tipo *float* o *int*. Es el mínimo número de muestras que tiene que haber en cada nodo para poder dividir, si hay menos de esa cantidad no podrá dividir más.
Min_samples_leaf	Puede tomar valores de tipo *float* o *int*, es el número mínimo de muestras para tener en cuenta un nodo hoja y seguir dividiendo por ese, si no cumple este parámetro ese nodo se omite. Por defecto toma el valor 1.
Min_weight_fraction_leaf	Toma valores de tipo *float*, se refiere al mínimo peso que puede tener cada hoja.
Max_features	Puede tomar valores de tipo int, float o auto, sqrt, log2. Se usa para buscar la mejor separación, en el parámetro *Splitter*.
Max_leaf_nodes	Número máximo de hojas hasta el que puede crecer un árbol. Toma valores enteros.
Min_impurity_decrease	También se relaciona con la división de los árboles, este nodo se divide si las impurezas generadas por la división son mayores que este valor.
Min_impurity_split	Un nodo se divide si su nivel de impurezas está por debajo de este valor.
Class_weight	Si se proporciona un diccionario, las claves van a ser las clases y los valores son los pesos. Si no se da ninguno, los pesos de la clase serán uniformes.
Ccp_alpha	Es un parámetro de complejidad que se usa en la poda para obtener el coste mínimo.

2.2.1.2. Bosques Aleatorios

El Bosque Aleatorio (Pedregosa et al. 2011) está compuesto de varios arboles de decisión independientes, los cuales tienden a especializarse en un determinado subconjunto del problema y se unen para dar un único resultado final representada en la Figura 2.3.

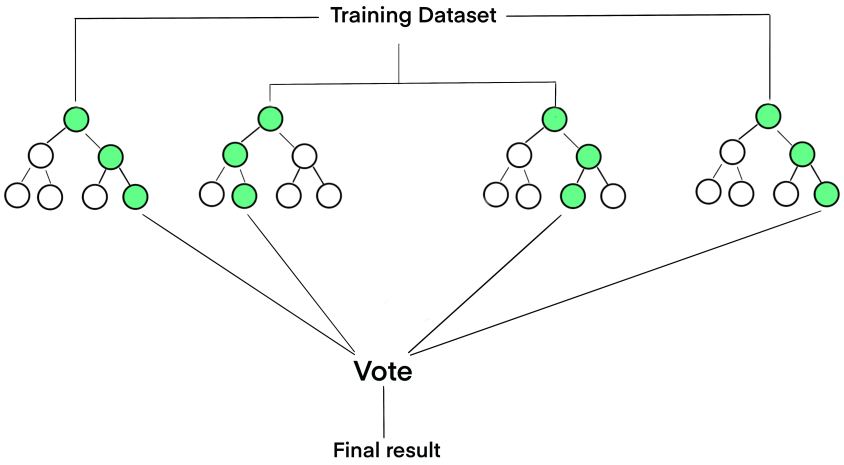


Figura 2.3: Representación de Random Forest

Este algoritmo se adapta con facilidad tanto a problemas de clasificación como problemas de regresión. Está compuesto por los parámetros que se muestran en la Tabla 2.2.

Tabla 2.2: Parámetros usados en el algoritmo de bosques aleatorios

Parámetro	Descripción
Number of estimators	Ss el número de Árboles de Decisión que conforman el Random Forest. Normalmente un mayor número de árboles resulta en mejores resultados pero con un crecimiento exponencial en los tiempos de ejecución. Después de cierto número de árboles los resultados apenas varían y podría llegar a sobreaprender.
Max_depth	Determina la profundidad máxima que puede tener el árbol. Al variar la profundidad varía el punto donde se podan los árboles.
Minimum_samples_split	Determina el número mínimo de ejemplos en un nodo del árbol para poder dividirse. Cuanto mayor sea más se agrupan los ejemplos, dando lugar a una mala generalización.
Minimum_samples_leaf	Define el número mínimo de ejemplos que deben quedar en las hojas. Al igual que en el *minimum_samples_ split*, un número alto puede conllevar a conjuntos muy grandes en las hojas y, por tanto, mala generalización.
Criterion	Métodos para medir cómo de buenas son las particiones de los nodos, podemos usar métricas como Gini o Entropía.

2.2.1.3. Máquina de soporte vectorial

Es un algoritmo de aprendizaje automático supervisado (del inglés *Support Vector Machine (SVM)*) que busca un hyperplano que divida los datos en diferentes grupos de acuerdo a sus caraterísticas. La Figura 2.4 muestra un ejemplo de hyperplano en 2 dimensiones que estaría representado por una recta (Pedregosa et al. 2011, Betancourt 2005). Se puede pensar que hay múltiples soluciones para dividir un plano con una recta, pero la máquina de soporte vectorial resuelve este problema cogiendo el punto de cada grupo más alejado, también llamado vector de soporte, e intentando maximizar su margen (distancia que lo separa de la recta que divide cada agrupación de datos), con

esto tendríamos solo una solución posible. Cabe destacar que podemos tener más de un vector de soporte por grupo, en ese caso se tendría que elegir el que mayor margen tenga. Idealmente una máquina de soporte vectorial va a separar los grupos de forma perfecta usando los hyperplanos mencionados anteriormente. Esto no siempre es posible, y si lo es, este modelo no puede ser usado con otro conjunto de datos ya que consideramos que ha sobreaprendido.

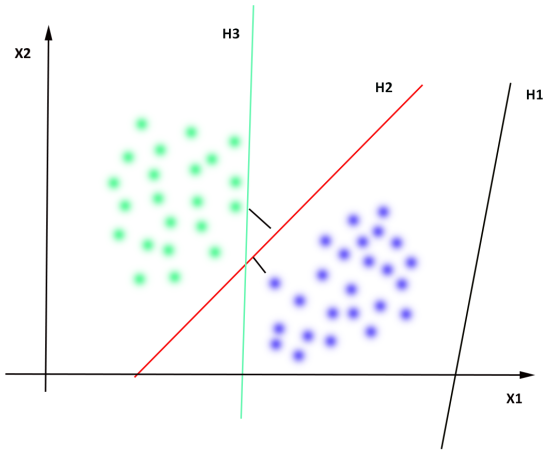


Figura 2.4: hyperplanos que dividen los dos grupos de puntos

Con el fin de contemplar un poco de flexibilidad, la SVM cuentan con un parámetro C, es un parámetro de regulación, es el único parámetro que puede ser ajustado a la hora de programar un clasificador con SVM. Este parámetro se encarga de regular como de precisos queremos que sea nuestra máquina, por lo tanto puede controlar cuanto sobreaprende. Alguna de las ventajas con las que cuenta este tipo de algoritmo clasificador es que se entrena de una forma muy sencilla. Además, no tenemos un óptimo local como en las redes neuronales, es un modelo escalable lo que nos permite usar distintos tamaños de conjuntos de datos con relativa facilidad.

2.2.1.4. Clasificador Bayesiano Ingenuo

Es un clasificador probabilístico basado en el teorema de Bayes. La fórmula del teorema de Bayes es la siguiente:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{2.3}$$

Sirve tanto para resolver problemas binarios, donde solo usamos dos clases, como para resolver problemas de clase múltiple. Se le conoce como Bayes Ingenuo ya que todas las probabilidades son simplificadas lo máximo posible para poder hacer una implementación que permita su exploración. La versión de Bayes Ingenuo más utilizada es Bayes Ingenuo Gaussiano aunque existen otras versiones que también se usan: Bayes Ingenuo de Bernouilli, el cual emplea una bolsa de palabras; Bayes Ingenuo Multinomial,

que es la versión no binaria de Bernouilli; y el Complement Naive Bayes, que es una adaptación del Multinomial, tal como se muestra en la Figura 2.5 (Webb 2010).

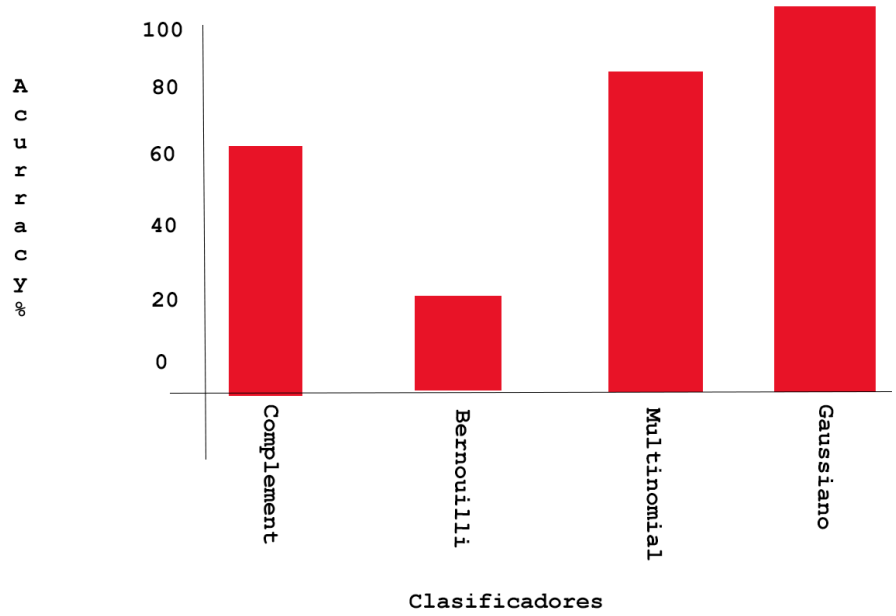


Figura 2.5: Comportamiento de las diferentes versiones de Naive Bayes

Los principales parámetros del Naive Bayes se muestran en la Tabla 2.3 (Pedregosa et al. 2011).

Tabla 2.3: Parámetros usados en el clasificador bayesiano ingenuo

Parámetro	Descripción
alpha	Parámetro de tipo float que determina el valor aditivo para suavizar los parámetros, con valor 0 no suaviza. Por defecto se suele usar valor de alpha igual a 1.
fit_prior	Parámetro booleano que determina si usamos las probabilidades de class_prior, en caso de ser falso se usará una prioridad uniforme.
class_prior	*Array* con las prioridades de las clases, por defecto está a *None*.

2.2.1.5. Regresión Lineal

Es uno de los algoritmos de aprendizaje supervisado más sencillos que hay (Heras 2020). Se dibuja una recta para encontrar los parámetros de esta que minimizan el error cuadrático medio de los datos, o lo que es lo mismo, queremos encontrar los mejores parámetros o coeficientes de la recta para los datos dados. Como entrada deberemos tener un conjunto de datos continuo. La fórmula de la recta con una sola variable la representada en la Ecuación 2.4.

$$Y = mX + b \tag{2.4}$$

donde, m es la pendiente y b el punto de corte con el eje x tal como se puede ver en la Ecuación 2.5.

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \epsilon \tag{2.5}$$

A ϵ lo consideramos como el error de nuestra predicción frente a la realidad, es una variable aleatoria.

Los parámetros que considera *sklearn* para regresión lineal según (Pedregosa et al. 2011) se muestran en la Tabla 2.4.

Tabla 2.4: Parámetros usados en el algoritmo de regresión lineal

Parámetro	**Descripción**
fit_intercept	Toma valores de tipo *bool*, por defecto toma el valor *true*. Se usa para calcular la intersección de este modelo. Si lo ponemos a falso, no se usarán intersecciones para calcular este modelo
normalize	Este parámetro es ignorado cuando *fit_intercept* se pone a *False*. Al igual que el anterior toma valores de tipo *bool*. Si está a *True* el modelo se normalizará
copy_X	Si toma el valor *True*, la x se copia, en otro caso se sobreescribe. *None*. Se refiere al número de procesos paralelos para procesar los cálculos.
positive	Cuando se pone a *True* se fuerza a los coeficientes a ser positivos. Por defecto es *False*.

2.2.1.6. Regresión Logística

Es un algoritmo de aprendizaje supervisado de los más sencillos y utilizados para la clasificación de dos clases. Se puede utilizar para diferentes problemas de clasificación. Este algoritmo lleva el nombre de la función que se utiliza, función logística o función sigmoide, previamente definida. Existen varios diferentes tipos de regresión logística: (González 2021*b*)

- **Binaria**: La variable de salida tiene dos posibles valores.
- **Multinomial**: La variable de salida tiene tres o más resultados nominal.
- **Ordinal**: La variable de salida tiene tres o más resultados ordinales.

Según scikit learn (Pedregosa et al. 2011) se pueden tomar los parámetros de la Tabla 2.5.

2.2.1.7. Aumento de Gradiente Extremo

Es una alternativa optimizada del algoritmo Potenciación del Gradiente, es más eficiente, rápido, flexible, y dispone de una librería de código abierto (xgboost developers 2020, Pedregosa et al. 2011). Dicho algoritmo implementa una serie de árboles de *boosting* en paralelo. Es muy eficiente, da muy buenos resultados y por eso ha ganado mucha popularidad en los últimos años. Una de sus mayores desventajas es la gran cantidad de memoria RAM que utiliza, por contra puede trabajar con grupos muy grandes de muestras. Los parámetros que utiliza Extreme Gradient Boosting (XGBoost) se muestran en la Tabla

2.6 (Rinfret 2019). Tanto *num_pbuffer* como *num_feature*, se ponen automáticamente, no necesitan intervención del usuario.

Tabla 2.5: Parámetros usados en el algoritmo de regresión logística

Parámetro	Descripción
penalty	Se refiere a la norma que se usa para la penalización. Puede tomar los valores: 'l1', 'l2', 'elasticnet', 'none'
dual	toma como valor *False* ya que la formula dual sólo se puede implementar si *penalty* tiene el valor 'l2'.
tol	Es la tolerancia para el criterio de parada.
C	Debe tomar un *float* positivo. Se refiere a la inversa de la fuerza de regularización.
fit_intercept	especifica si una constante se debe añadir a la función de decisión.
solver	Son los algoritmos usados para el problema de optimización. Puede tomar los valores: 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'.
intercept_scaling	Se utiliza cuando el *solver* toma como valor 'liblinear'. Toma valores de tipo *float*.
class_weight	Se refiere a los pesos asociados a cada clase.
max_iter	Es el número máximo de iteraciones que puede toner el *solver* para converger. Por defecto se toma el valor 100.
multi_class	Si toma el valor 'ovr' cada etiqueta se ajusta a un problema binario. También puede tomar los valores 'auto' y 'multinomial'.
verbose	Valor entero positivo para determinar el nivel de mensajes del *liblinear* y del *lnfgs*.
warm_start	Valor booleano que cuando está puesto a *True* usa la solución de la última llamada para servir de inicialización. En caso de ser *Fase* simplemente borra las anteriores soluciones.
l1_ratio	valor comprendido entre 0 y 1. Solamente usado cuando el parámetro *penalty* tiene el valor 'elasticnet'. Cuando tiene el valor 0 equivale a usar penalty con 'l2', cuando toma el valor 1 equivale a usar 'l1' en el *penalty*, y los valores intermedios sirven para usar una combinación de ambos como penalización.

Tabla 2.6: Parámetros usados en el algoritmo XGBoost

Parámetro	Descripción
Booster	Indica el tipo de aprendizaje a utilizar, por defecto se selecciona 'gbtree' pero puede tomar los valores: 'gbtree', 'gblinear' y 'dart'.
verbosity	Se refiere a la calidad de los mensajes que vamos a imprimir, los valores que puede tomar son *(0)silence*, *(1) warnign*, *(2) info* y *(3) debug*.
validate_parameters	Puede tomar los valores *True* o *False*. Cuando ponemos que es *True* XGBoost va a validar los parámetros de entrada para verificar si un parámetro se usa o no.
nthreads	Número de hilos paralelos que utiliza XGBoost.
disable_default_eval_metric	Se usa para desabilitar la métrica por defecto.
num_pbuffer	El tamaño del *buffer* de predicción.
num_feature	El número de características que va a usar XGBoost.
N_estimators	Acota el número de estimadores que va utilizar.
Learning_rate	Se refiere a la forma en la que se ajjustan los pesos.

2.2.1.8. K-Vecinos más Cercanos y K-Vecinos más cercanos con pesos

El algoritmo KNN es un algoritmo simple de clasificación. El algoritmo consiste en dos pasos: (Wu et al. 2018)

1. Para encontrar los K-Vecinos más cercanos para un elemento nuevo A que se introduzca, se utiliza una métrica de distancia D (generalmente la euclídea). Se busca qué elementos de la muestra de datos son los que están a menor distancia del nuevo elemento. En el caso de la distancia euclídea, se comparan todos los n parámetros del elemento A ($a_{1..n}$) con todos los n parámetros de cada elemento B ($b_{1..n}$) que ya estuviesen dentro de los datos tal como se muestra en la Ecuación 2.6.

$$D = \sqrt[2]{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{2.6}$$

2. A partir de estas distancias se toman los k elementos más cercanos y se mira cual es su clase. La clase que tenga el mayor número de elementos dentro de los k más cercanos es la clase que se asignará al nuevo elemento A, es decir, la moda.

Los parámetros que utiliza KNN se muestran en la Tabla 2.7 (Pedregosa et al. 2011).

Tabla 2.7: Parámetros usados en el algoritmo K vecinos más cercanos

Parámetro	**Descripción**
n_neighbors	Se refiere al número de vecinos que se va a usar, por defecto toma el valor 5.
weights	Es el peso que se va a usar para hacer la predicción. Puede tomar los valores: *'uniform'* en el que todos los vecinos se pesan igual o *'distance'* donde los pesos son el inverso de la distancia, en este caso los vecinos más cercanos tendrán más peso.
algorithm	Es el algoritmo que se usará para determinar cuales son los vecinos más cercanos. Por defecto tomará el valor *'auto'* que determinará el algoritmo más apropiado para cada situación, pero se lo podemos cambiar por: *'ball_tree'*, *'kd_tree'*, *'brute'*.
leaf_size	Es tamaño de las hojas que se le pasa en caso de que usemos el algoritmo *ball_tree*.
metric	Se refiere a como se medirá la distancia usada en los árboles, podemos usar distancias como la Euclídea o Minkowski, que es la que se usa por defecto.
metric_params	parámetros adicionales para la función de métrica.

El algoritmo *weighted* KNN, es una variante del algoritmo KNN que aporta unos *pesos* a los vecinos según su distancia. Estos pesos hacen que no solo cuenten las apariencias de elementos de clase dentro de los k-elementos cercanos escogidos, sino que además importa cómo de cerca estén. Cuanto más cerca estén más peso tendrán. Para medir la distancia y los pesos se pueden usar diferentes métricas, la más usada es la distancia euclídea como distancia y la inversa al cuadrado de la distancia como peso. Es decir, tal como se muestra en la Ecuación 2.7, se toma como función la suma de las inversas de la distancia entre el nuevo elemento y todos los elementos de una clase j.

$$Sum_j = \sum_{i=1 \cap i \in C_j}^{n} \frac{1}{D_i} \tag{2.7}$$

2.2.2. Técnicas de Aprendizaje no Supervisado

Como se indica en la sección 2.2 el aprendizaje no supervisado se caracteriza por no utilizar conocimiento previo sobre un conjunto de datos para etiquetarlo, infiriendo patrones para esta tarea. Se puede utilizar para descubrir la estructura subyacente de los datos. Uno de los algoritmos más utilizados de este tipo es k-medias.

2.2.2.1. K-medias

Clustering: (Ollé n.d.) Los *clusters* son un tipo de algoritmos de aprendizaje automático no supervisado. Se basan en agrupar individuos de nuestro conjunto de datos teniendo en cuenta algunas características comunes. Los puntos que estén en el mismo grupo deben tener propiedades similares.

K-medias es un algoritmo de aprendizaje no supervisado que utiliza clusters. El algoritmo utiliza tres pasos (Universidad de Oviedo 2021):

1. Inicialización: se elige el número de grupos que queremos clasificar, la k, y se establecen ese mismo número de centroides en ese mismo conjunto de datos, se pueden escoger de forma aleatoria.

2. Asignación de los objetos a los centroides: cada objeto de los datos es asignado a su centroide más cercano.

3. Actualización de los centroides: se actualiza la posición de los centroides de cada grupo, se toma como nuevo centroide la posición promedio de los objetos de dichos grupos.

Los pasos 2 y 3 se repiten hasta que los centroides no cambian o se mueven por debajo de una distancia umbral que se establece. Con el algoritmo de k-medias se resuelve un problema de optimización siendo la función a optimizar la suma de las distancias cuadráticas de cada objeto al centroide de su *cluster* (Pedregosa et al. 2011). Los parámetros que utiliza KNN se muestran en la Tabla 2.8.

Tabla 2.8: Parámetros usados en el algoritmo K medias más cercanos

Parámetro	Descripción
n_clusters	Se refiere al número de *clusters* que se va a usar, siendo este el número de centroides que se tienen que obtener.
Init	Es el método que se usa para la inicialización de los centroides, puede ser aleatorio o de una forma inteligente que permita una convergencia rápida.
n_init	Es el número de veces que se ejecutará el algoritmo con diferentes semillas para los centroides.
Max_iter	Es el número máximo de iteraciones que podrá hacer el algoritmo cada vez que se ejecute.
tol	Se refiere a la tolerancia relativa con respecto a la norma de Forbenius.
Verbose	Nivel de verbosidad.
precompute_distances	
Copy_x	Valor booleano puesto por defecto a *True*. TODOOOO
n_jobs	Se refiere al número de hilos que se van a usar. simultaneamente
Algorithm	El algoritmo que va a usar K medias, puede tomar el valor 'auto', 'full'y 'elkan'. Este último algoritmo es más eficiente si los *clusters* están bien definidos.

2.2.3. Aprendizaje Profundo

Los algoritmos de aprendizaje profundo pueden acceder a un mayor número de datos por lo que llegan a conseguir, no solo que la máquina aprenda, sino que también consiga experiencia. Una vez que estas máquinas tienen la suficiente experiencia, podrán realizar tareas tales como conducir un coche, detectar enfermedades, etc. (NetApp 2021). Como principal algoritmo usa las redes neuronales debido a su gran capacidad por procesar datos y ya que lo puede hacer en paralelo.

2.2.3.1. Red Neuronal

La red neuronal es una estructura de datos que aprende mediante iteraciones a qué variables hay que darle mayor importancia a la hora de clasificar (Goldberg 2017, Pedregosa et al. 2011). Se puede ver la red neuronal como una tupla $(X, W, \sum, f, Y)$ donde:

- X es el conjunto de entradas. $(x_1...x_n)$
- W es el conjunto de pesos. $w_{i,j,k}$ donde i es el numero de arista entrante de la neurona j en la capa k
- $\sum$ es la función de agregación.
- f es la función de activación.
- Y es la salida.

En este caso las entradas son el estímulo al que se sometería la red neuronal, y las salidas, la respuesta ante dicho estímulo. Podemos ver la red neuronal como un grafo en

el que cada uno de los nodos de una capa está conectado con todos los nodos de la capa siguiente y la anterior. El conjunto de entradas $(x_1...x_n)$ serán los parámetros que podamos extraer de la muestra.

Por otro lado, el conjunto de pesos de la neurona $(w_{1,1}...w_{1,n}...w_{m,n})$ nos indicará como de importante es la salida de una neurona de la capa anterior para entrada de la neurona de la capa siguiente. Habrá un peso w por cada arista del grafo. Cada neurona tendrá una salida mediante la cual enviará información a las neuronas de la capa siguiente, o a la salida de la red neuronal, en el caso de que se trate de la última capa de neuronas.

Para decidir la salida de una neurona i de una capa, se aplica la función de agregación de la Ecuación 2.8, es decir, el sumatorio de las salidas de las neuronas anteriores multiplicadas por los pesos. En el caso de la capa inicial, la función de agregación $(\sum)$ será la entrada.

$$\sum(j,k) = w(1,j,k) * \sum(1,k-1) + w(2,j,k) * \sum(2,k-1) + ... + w(n,j,k) * \sum(n,k-1) \quad (2.8)$$

A este resultado se le sumará un *bias*, y se le aplicará una función de activación, la cual nos dará el resultado final de esa neurona (Misra 2019):

- **Funcion de umbral**: Las funciones de umbral de la Ecuación 2.9 computan una señal de salida distinta dependiendo o no de si su entrada supera un cierto umbral. Los valores de entrada para una función de activación es la suma (aplicando los pesos) de los valores de entrada de la capa anterior. Una buena definición de la función de umbral, en este caso suponiendo $umbral = 0$ sería la siguiente:

$$F(x) = \begin{cases} 0 \text{ si } x \leq 0 \\ 1 \text{ si } x > 0 \end{cases} \quad (2.9)$$

- **Función Sigmoide**: La función sigmoide computa un valor para la señal de salida entre 0 y 1 dada una entrada cualquiera. Por otra parte, tiene la propiedad de suavidad, lo cual hace que admita derivadas de cualquier orden en cualquier punto de la función aunque en sus extremos tenderá a ser 0. Otro uso que se le da a la función sigmoide con Ecuación 2.10 en aprendizaje supervisado es la regresión logística.

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (2.10)$$

- **Función Rectified Linear Unit (ReLU)**: La función ReLU no tiene la propiedad de suavidad como la sigmoide. Pese a esto, sigue siendo una opción muy popular dentro del campo del aprendizaje profundo.

$$F(x) = max(x, 0) \quad (2.11)$$

- **Función de la Tangente hiperbólica**: Función basada en la identidad trigonométrica según la Ecuación 2.12, tiene la propiedad de suavidad y es derivable en todos sus puntos, aunque tiende a 0 en sus extremos.

$$\phi = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.12}$$

La red neuronal que podemos ver en la Figura 2.6 irá ajustando los bases y los pesos de cada una de las neuronas, y las aristas, respectivamente, aplicando una función de coste que irá reduciendo con el algoritmo del descenso del gradiente estocástico. Cabe además destacar que este clasificador es un modelo de caja negra. Es decir, se hace muy dificultoso explicar por qué una red neuronal ha tomado una decisión.

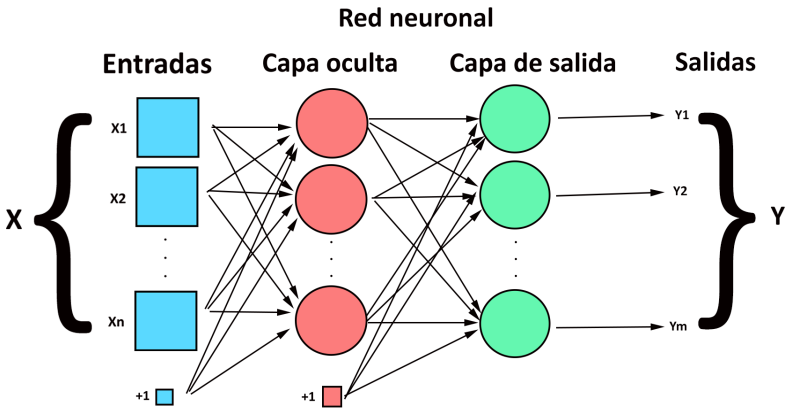


Figura 2.6: Representación de una red neuronal

2.3. Métricas Utilizadas en Problemas de Clasificación

Cuando se está evaluando cómo de buena va a ser una clasificación, tenemos métricas que nos ayudan en este proceso. Se clasifican de la siguiente forma:

- **Precision (Precisión)**: Con la precisión se puede medir la calidad del clasificador. Se refiere al número de veces o porcentaje que el clasificador acierta. Pongamos un ejemplo muy usado en este campo para explicarlo. Se tiene un clasificador que analiza tumores, este clasificador tiene una precisión de 0.5, eso quiere decir que cuando predice que un tumor es maligno acierta el 50 % de las veces. Para medir la precisión se usa la Ecuación 2.13.

$$precision = \frac{VP}{VP + FP} \tag{2.13}$$

- ***Accuracy* (Exactitud)**: Mide el porcentaje de casos que el modelo acierta. Con el mismo ejemplo de los tumores podemos decir que si nuestro clasificador nos da un

70 % de exactitud, el modelo acierta el 70 % de las veces. Esta métrica no funciona bien cuando tenemos las clases desbalanceadas. La fórmula que usa el *accuracy* es la Ecuación 2.14.

$$accuracy = \frac{VP + VN}{VP + FP + VN + FN} \tag{2.14}$$

- ***Recall* (Exhaustividad)**: Intenta responder a la pregunta ¿Que porcentaje de Verdadero Positivo (VP) se identificó correctamente? La exhaustividad y la precisión suelen estar inversamente relacionadas por lo que, normalmente, si una aumenta la otra disminuye. Usa la Ecuación 2.15.

$$exhaustividad = \frac{VP}{VP + FN} \tag{2.15}$$

- **F1**: Se usa para combinar las medidas de precisión y *recall* en un solo valor. Es una medida muy útil para poder medir el rendimiento de nuestro modelo. La fórmula de F1 es la Ecuación 2.16.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.16}$$

  F1 asume que nos importa lo mismo precisión y *recall*, pero no siempre tiene por que ser así, por eso se usa la medida Fβ, dándole más importancia a la exhaustividad o a la precisión según convenga. La fórmula para Fβ es la Ecuación 2.17.

$$F\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \tag{2.17}$$

  Se sustituye β por el número de la F, por ejemplo con F2 se sustituye β por 2.

Para tomar estas métricas, además de para tener una visualización rápida de los resultados, se utilizan las llamadas Matrices de confusión.

Con la matriz de confusión expuesta en la Tabla 2.9 se puede ver de una forma más gráfica como se han clasificado los datos. Si se tiene dos etiquetas, la matriz de confusión va a dividir los resultados en: VP, Verdadero Negativo (VN), Falso Positivo (FP) y Falso Negativo (FN), obteniendo así una matriz de 2x2. Los VP son aquellos resultados que el clasificador ha etiquetado como positivo y realmente si lo son, lo mismo pasa con VN, por lo tanto el clasificador los ha etiquetado bien. Por otro lado tenemos los FP y los FN, estos hacen referencia a los que el clasificador ha dicho que son positivos y son negativos o al revés, por lo tanto están mal etiquetados. La matriz de confusión puede tener más de dos opciones por lo que pasaría de ser una matriz de 2x2 a ser una de 3x3 si son 3 opciones, 4x4 si son 4 (Arce 2019*a*).

Tabla 2.9: Matriz de confusión

Precition / Class	0	1
0	VN	FP
1	FN	VP

2.4. El problema del Sobreaprendizaje

En el aprendizaje supervisado, aquel que se basa en etiquetas de clasificación para entrenar los datos, el principal problema es el sobreaprendizaje u *overfitting*. Un modelo sobreaprendido es aquel que rinde muy bien cuando trabaja sobre el *dataset* con el que ha sido entrenado pero tiene un rendimiento muy pobre a la hora de predecir un nuevo *dataset*. También existe el caso contrario, el subaprendizaje o *underfitting*, donde nuestro modelo es demasiado simple y generaliza más de lo que debería, no aprende del *dataset* y por tanto no predice bien. Para que el modelo se considere un buen clasificador se tiene que encontrar el punto medio entre el sobreaprendizaje y subaprendizaje (Dietterich 1995) tal como se ve en la Figura 2.7.

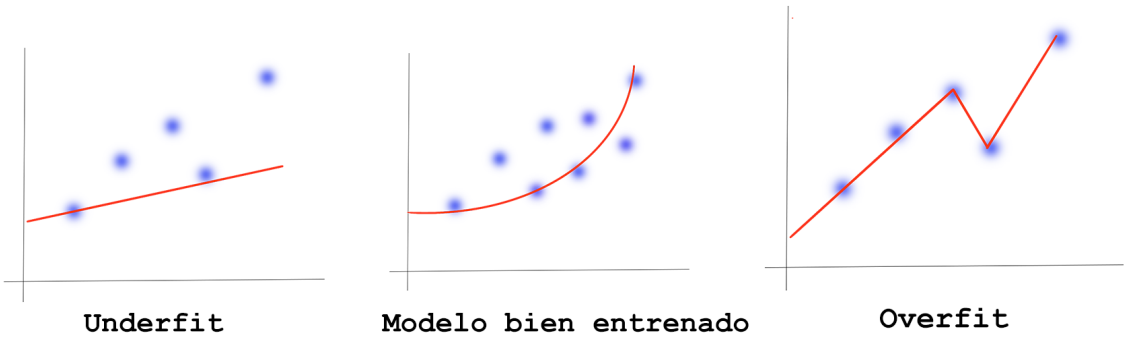


Figura 2.7: Gráficas que muestran el subaprendizaje, sobreaprendizaje y un modelo bien entrenado

Las dos métricas que se usan para medir el sobreaprendizaje son la varianza y el *bias*. La varianza en aprendizaje supervisado es la diferencia entre lo bien que ajusta los datos de entrenamiento y los de test. Al contrario de lo que se puede pensar, cuanto más alta es la varianza más ha sobreaprendido el modelo. Según el tipo de algoritmo, el modelo es más susceptible de sobreaprender o de tener una alta varianza (Sanmartín 2020). El *bias* o sesgo es la diferencia entre la predicción de nuestro modelo y los valores reales. Un bajo *bias* quiere decir que hay menos suposiciones sobre la función objetivo. En cuanto al *bias* se refiere también se debe tener en cuenta la complejidad del algoritmo. Algoritmos simples como la regresión lineal tendrán un *bias* alto, es decir, no ajustarán bien modelos que tienen patrones no lineales, sin importar el número de observaciones del *dataset*. A esto se le llama ajuste insuficiente (González 2021*a*).

El objetivo de cualquier algoritmo es conseguir tanto un *bias* como una varianza baja, a la vez que un buen rendimiento predictivo. Además estas dos medidas son inversamente proporcionales, si disminuimos el *bias* aumentará la varianza y viceversa por lo que encontrar el equilibrio óptimo es la clave.

2.4.1. Cómo Prevenir el Sobreaprendizaje

Por suerte existen muchos mecanismos para reducir el sobreaprendizaje, estos son los más usados (EliteDataScience 2017):

- **Validación cruzada**: Consiste en dividir el conjunto de entrenamiento en varias partes, usando diferentes subconjuntos para entrenar y comparar las predicciones con el otro subconjunto nunca visto antes por el algoritmo, de este modo podremos observar cuando el algoritmo está sobreaprendiendo. Dentro de la validación cruzada estas son las principales estrategias (Browne 2000).
  - *Single holdout method*: Consiste en excluir aleatoriamente casos del *dataset* para usarlos posteriormente como *test*, normalmente se excluyen entre el 10 % y el 30 %. Este método se usa cuando el *dataset* es suficientemente grande, en cuyo caso se podrá medir de forma fiable el error real del modelo ante casos no vistos.
  - *k-fold random subsampling*: Se repite el *single holdout method* k veces, de modo que obtenemos k pares $D_{train,j}$ y $D_{test,j}$ de modo que los conjuntos de un mismo par son disjuntos pero pueden solaparse los diferentes conjuntos de test entre sí. Se usa la media de los resultados para evaluar el error real del modelo.
  - *K-folds cross-validation* consiste en dividir aleatoriamente el *dataset* en k subconjuntos disjuntos e iguales en tamaño. El modelo se entrena con k-1 subconjuntos y se usa el restante para medir su desempeño. Este proceso se repite cambiando el subconjunto de test hasta haber pasado por todos, es decir, se repite k veces. A diferencia del *k-fold random subsampling*, en este caso los conjuntos de test son disjuntos entre sí, y el número de iteraciones depende del tamaño de los conjuntos de test. Este método es usado sobre todo cuando se tiene un *dataset* pequeño.
  - *Leave-one-out cross-validation* es un caso especial del *k-folds cross-validation* donde la k es igual al número total de muestras, de modo que cada vez dejamos un solo elemento fuera para hacer el test. En este caso obtenemos obtenemos el error real apenas tiene sesgo, sin embargo tendrá una varianza alta ya que los conjuntos de test difieren en un solo elemento. El coste computacional en este caso es muy alto para *datasets* muy grandes.

- **Aumentar el dataset para entrenar el modelo**: Aunque no es aplicable en todos los casos, en muchos casos aumentando el tamaño el conjunto de entrenamiento de un modelo conseguimos que el algoritmo tenga una mayor oportunidad de aprender factores claves a la hora de hacer predicciones. Se debe tener cuidado de no aumentar el ruido del *dataset* a la hora de aumentar su tamaño, pues esto únicamente perjudicará sus resultados. Por eso es importante asegurarse de que los ejemplos que componen el *dataset* son correctos y relevantes.

- ***Early stoping***: Consiste en parar de entrenar a nuestro algoritmo antes de que sobreaprenda. Se aplica a los modelos que iteran varias veces sobre el conjunto de entrenamiento para aprender. Después de cada iteración se mide su capacidad de generalizar, y en caso de haber mejorado, se seguiría iterando hasta un número máximo de iteraciones o hasta que se detecte sobreaprendizaje.

Capítulo 3

Procesamiento de Lenguaje Natural y Análisis de sentimientos

Para seguir con la práctica el análisis de llamadas hay que comprender en que consiste el lenguaje natural y como se lleva a cabo su procesamiento además de definir de forma teórica el análisis de sentimientos. En la Sección 3.1 se muestra la definición de lenguaje natural así como sus niveles de análisis. Seguidamente, en la Sección 3.2 se presenta una introducción al procesamiento computacional del lenguaje. En ella se exponen los conceptos que se van a usar en dicho procesamiento así como de las aplicaciones. Finalmente y dentro de esta misma sección, en concreto la sección 3.2.3, define lo que es el análisis de sentimientos, sus aplicaciones y antecedentes para terminar exponiendo su funcionamiento.

3.1. Definición de Lenguaje Natural

Lenguaje Natural (LN) es la forma de comunicación que usan las personas, de un modo cotidiano, para entablar conversación con otras personas. Con el tiempo esta forma de comunicación se ha ido perfeccionando a partir de la experiencia de tal forma que se puede llegar a analizar situaciones realmente complejas y con ello poder razonarlas. Teniendo en cuenta la sintaxis, un LN puede ser modelado por un Lenguaje Formal (LF), muy parecidos a los que se usan en matemáticas o lógica (Cortez Vasquez et al. 2009).

El LN va a tener 2 objetivos principales definidos según la Inteligencia Artificial (IA) (Cortez Vasquez et al. 2009):

- **Objetivo 1:** Facilitar a los usuarios que no estén especializados con los ordenadores una comunicación más sencilla entre ambos.

- **Objetivo 2:** Diseñar sistemas que puedan realizar tareas más complejas dentro del ámbito lingüístico, como pueden ser resúmenes de textos, traducciones... y para ello se va a modelar procesos cognitivos.

En la Figura 3.1 se muestra un esquema del funcionamiento del LN el cual se va retroalimentando a través de unas determinadas reglas gramaticales que son usadas en la forma de comunicación de las personas.

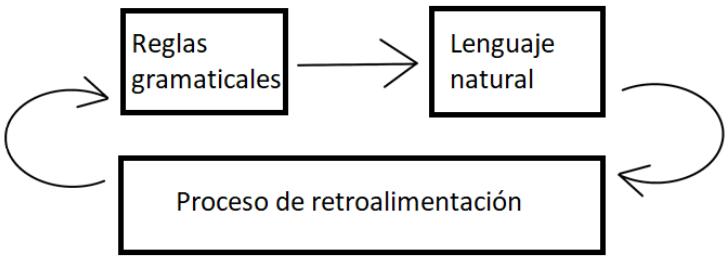


Figura 3.1: Esquema de comunicación (Cortez Vasquez et al. 2009)

3.1.1. Niveles de Analisis en el Procesamiento del Lenguaje Natural

El sistema PLN tiene que realizar una serie de tareas que faciliten la comunicación entre el usuario y el propio sistema. Con todas estas tareas se llegará a una arquitectura de niveles, donde las oraciones que se van a analizar puedan ser comprendidas por el sistema PLN.

Vamos a tener 4 niveles de análisis, tal como se muestra en la Figura 3.2, pero no todos estos niveles serán implementados (Cortez Vasquez et al. 2009, Sobrino Sande 2018). Esto se debe a que dichos niveles se van a ir desarrollando o no dependiendo de las funciones que tenga que desarrollar un determinado sistema:

- **Nivel de análisis fonológico:** nivel en el cual las palabras se relacionan con los sonidos que estas puedan representar.
- **Nivel de análisis morfológico:** nivel donde las palabras son examinadas morfológicamente y donde se va a extraer prefijos, sufijos y demás elementos. Este nivel tiene como principal objetivo ver como una palabra se va construyendo a partir de unos elementos más pequeños llamados *morfemas*.
- **Nivel de análisis sintáctico:** nivel que analiza cómo se estructura una oración teniendo en cuenta un determinado modelo gramatical empleado.
- **Nivel de análisis semántico:** nivel que otorga un significado a una oración y para ello resolverá cualquier ambigüedad léxica que pudiera aparecer en dicha oración.
- **Nivel de análisis pragmático:** nivel que analiza un texto como tal formado por varias oraciones. Se va a tener en cuenta también aquellas oraciones anteriores a una actual que se esté analizando para ver la relación que pudiera haber entre ellas.

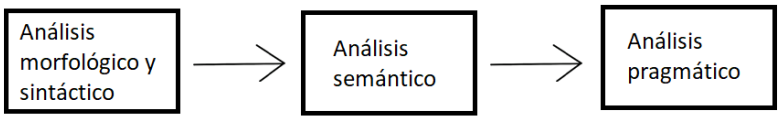


Figura 3.2: Niveles de Análisis (Cortez Vasquez et al. 2009)

3.2. Procesamiento Computacional del Lenguaje Natural

El PLN es un área de la Inteligencia Artificial, se basa en usar un lenguaje natural que se comunique con el ordenador; y para ello dicha máquina tiene que entender las oraciones que le pueda proporcionar. El uso de estos lenguajes va a desarrollar modelos que faciliten la comprensión de mecanismos relacionados con el lenguaje (Cortez Vasquez et al. 2009).

3.2.1. Conceptos del Procesamiento del Lenguaje Natural

A continuación se van a mostrar de forma breve los distintos conceptos que se van a utilizar en el procesamiento del lenguaje natural.

- **N-Grama**: Un n-grama es una subsecuencia de n elementos de un conjunto, por ejemplo podemos coger subsecuencias de palabras de una frase. Estos n elementos se cogen de forma no necesariamente correlativa, pero para el PLN, se entiende que son elementos contiguos.Si tenemos 1-grama se llama unigrama, si tenemos 2-grama se llama bigrama o diagrama y si tenemos 3-grama se llama trigrama (EcuRed 2019, de los Santos 2020).

- ***Tokens***: Un *token* es una instancia de una secuencia de caracteres pertenecientes a un documento que se agrupan como una unidad semántica para ser procesadas. Para esto se utiliza un proceso llamado Tokenización (Manning et al. 2009).

- **Bolsas de palabras**: Es una forma de representar el vocabulario de nuestro *corpus*, consiste en crear una matriz donde cada columna es un *token* y las filas son las frases que tiene nuestro corpus, dentro de nuestra matriz irá el número de veces que aparece cada token en cada oración (Soto 2018).

- **Preprocesado del texto**: El preprocesado es un paso fundamental a la hora de PLN, se trata de darle la forma que necesites a tu *corpus*, puede decidir si quitar o dejar signos de puntuación, si darle peso a las mayúsculas, si filtrar *stopwords*, que son palabras "vacías" como artículos, pronombres, preposiciones... Existen varias técnicas para el preprocesado del *corpus* (Mayo 2018):

  - **Lematización**: Tiene en cuenta el análisis morfológico de la palabra, para esta tarea es necesario el uso de un diccionario para poder buscar en él el lema. Por ejemplo, de estudiar y estudiando, el lema sería estudia (Soto 2018).

- ***Stemming***: Se quita el final o el principio de las palabras teniendo en cuenta una lista de prefijos y sufijos quedándose únicamente con el *stem*. Por ejemplo de niño y niñera el stem será niñ. Este proceso no siempre corta bien las palabras por lo que es un algoritmo que presenta limitaciones (Soto 2018).

Una vez hecho el preprocesado del texto se necesita un sistema de ponderación numérica, los dos sistemas de ponderación más relevantes son:

- **TF**: Muestra cómo de frecuente es un término en un documento representando en la Ecuación 3.1.

$$tf(t,d) = \frac{f(t,d)}{(\text{máx}\, f(t,d) : t \in d))} \tag{3.1}$$

- **TF- IDF**: Se compone de dos partes, en primer lugar se usa TF que muestra como de frecuente es un término en un documento, por otro lado se encuentra IDF representando en la Ecuación 3.2 que se refiere a la frecuencia inversa de un documento, esto se utiliza para reducir el peso de un término mostrado en la Ecuación 3.3, si esta aparece en muchos documentos (Soto 2018).

$$idf(t,D) = \log \frac{|D|}{|d \in D : t \in d|} \tag{3.2}$$

$$tfidf(t,d,D) = tf(t,d) * idf(t,D) \tag{3.3}$$

3.2.2. Aplicaciones del Procesamiento del Lenguaje Natural

Vamos a detallar algunas de las aplicaciones del PLN (Pauli 2019):

- **Traducción automática de textos:** Constituye unas de las aplicaciones más importantes del PLN. Para una persona corriente la labor de traducir textos resulta una tarea complicada ya que no es fácil encontrar a alguien que conozca varias lenguas, o una combinación de ellas (por ejemplo mezclar el ruso y chino simultáneamente). Los sistemas de traducción automática que hay en la actualidad usan un determinado enfoque que realiza mediciones estadísticas y relacionar textos. Para ello se llevará a cabo previamente un entrenamiento con un gran número de textos. Aunque estas traducciones no siempre son perfectas, si son aceptables para aquellas destinadas a traducir un mensaje procedente de una red social, reseña o pagina web. Uno de estos ejemplos es el *Google Translate*.

- **Recuperación de la información:** Con esta aplicación se va a conseguir la obtención de un grupo de documentos electrónicos a través de un conjunto de palabras clave que hayan aportado los usuarios. Estos documentos se van a ordenar teniendo en cuenta un atributo que medirá su relevancia dentro del resultado final.

Un ejemplo de este tipo de sistema son los que realizan los buscadores de contenidos de internet y van a constituir la primera aplicación que se ha implantado dentro de las Tecnologías de la Información. *Google Search* o *Microsoft Bing* son algunos de estos ejemplos.

- **Extracción de Información y Resúmenes:** Esta tarea consiste en analizar mensajes o textos con el fin de extraer información que tenga algún interés. Para ello se van a escanear documentos escritos en LN y después toda esa información se extrae a una base de datos. Algunos ejemplos de datos que nos pueden interesar son teléfonos, fechas, etc...

- **Reconocimiento de Voz:** Son sistemas que permiten a las personas interactuar con dispositivos electrónicos haciendo uso del lenguaje natural y la voz. Algunos ejemplos, muy conocidos por todos, son los llamados "asistentes virtuales", como *Alexa* o *Siri* que pueden llevar a cabo acciones como realizar la compra, enviar un correo electrónico o encenderte la luz, con solamente el uso de la voz. Otro ejemplo de uso son los servicios de atención al cliente usados en telefonía, que pueden realizar diferentes gestiones y acciones sin necesidad de una persona humana y usando una máquina.

- **Tutores inteligentes:** Son sistemas que actúan como un tutor particular de un determinado estudiante. Debido a esto el sistema tiene que tener la libertad para actuar dependiendo de las necesidades del estudiante. Se quiere diseñar un sistema que este adaptado según la capacidad de cada estudiante y de acuerdo a unos conocimientos previos.

- **Análisis de sentimientos:** Son sistemas que determinan el tono emocional que puede haber detrás de una cantidad de palabras. Con todo esto se puede entender que opiniones o emociones son expresadas por los usuarios sobre un determinado producto.

3.2.3. Definición de Análisis de Sentimientos

El concepto de análisis de sentimiento (AS o SA por sus siglas del inglés *Sentiment Analysis*) hace referencia a los distintos tipos de lingüística computacional que van a ayudar a extraer una información subjetiva de un contenido digital como por ejemplo foros, *webs*, redes sociales, etc. Vamos a poder extraer de un texto de internet un valor que sea tangible, determinando qué connotaciones positivas o negativas tiene dicho texto. Se trata de un campo incluido en el PLN que extrae información subjetiva de un texto a través de técnicas computacionales (Sobrino Sande 2018).

En la actualidad, todos los usuarios tienen una gran facilidad para exponer sus opiniones sobre un determinado tema. Poder tener una constancia de las opiniones relacionadas con un producto o una marca, es vital para las empresas, ya que con esto se forma una "minería de opinión", del que las empresas pueden hacer uso de forma inmediata. La investigación del análisis de sentimientos alcanza su mayor auge a partir del 2001, incrementándose con el paso de los años. Esto se debe principalmente a 3 factores : (i) La forma en la que se han popularizado los métodos de aprendizaje y su uso en PLN; (ii) La disponibilidad que se tiene a los datos con los que entrenar haciendo uso del sistema de Internet; (iii) gran interés que aumenta cada día por parte de empresas, para obtener una valoración de sus productos llevada a cabo por los usuarios (Sobrino Sande 2018).

3.2.3.1. Aplicaciones del Análisis de Sentimientos

Se va a distinguir las siguientes aplicaciones dentro del AS. (Pauli 2019):

- **Valoración de la opinión en determinados servicios y productos:** Esta aplicación es probablemente la más utilizada en análisis de sentimientos. Con ella las empresas pueden saber que opinión tienen los usuarios sobre unos determinados productos sin tener que realizar previamente estudios como por ejemplo "encuestas de satisfacción". Es posible conocer lo que les gusta a las personas y plantear estrategias en caso de que un producto no sea del agrado de las personas.
- **Posicionar publicidad *online*:** Con esta aplicación los anunciantes de un producto pueden publicar en sitios web donde se expresen conceptos positivos de dicho producto.
- **Mejorar sistemas que hagan una recomendación de un producto:** Una tienda *online* podrá destacar productos dependiendo de las opiniones que tengan los usuarios y no recomendar los productos que no sean de su agrado.
- **Corregir una opinión:** Hay muchos sitios de compra *online* en el cual los usuarios indican tanto su opinión con una reseña como con una puntuación. Un sistema de análisis de sentimientos podría analizar las palabras de un usuario y corregir la puntuación, en caso de que no se haya indicado correctamente.
- **Analizar el mercado financiero:** Se puede preveer como será la evolución de un mercado financiero a partir de la información que haya en foros o *webs*.
- **Reputación política:** Esta aplicación es una gran potencial con el que se puede conocer la opinión que tiene la gente sobre un candidato o partido político.

3.2.3.2. Antecedentes del Análisis de Sentimientos

En los últimos años el análisis de sentimiento se ha convertido en un tema de investigación jerárquico debido a la gran demanda existente en el mercado por analizar

la opinión pública. Sus orígenes se remontan al año de 1996, con el desarrollo del sistema General Inquirer, el cual agrupa las palabras en diversas categorías (Stone & Smith 1966). Con el paso del tiempo han surgido nuevas técnicas, librerías y herramientas para llevar a cabo el PLN. En la Tabla 3.1 se presentan algunas de estas herramientas y librerías.

Tabla 3.1: Herramientas y librerías para PLN

Nombre	Tipo	Uso	Referencia
FreeLing	Librería	Libre	(Carreras et al. 2004)
TextBlob	Librería	Libre	(Gujjar & HR 2021)
NLTK	Kit de herramientas Libre	Libre	(Loper & Bird 2002)
OpenNLP	Librería	Libre	(Kottmann et al. 2011)
Quanteda	Librería	Libre	(Benoit et al. 2018)
SentiWordNet	Léxico	Libre	(Esuli & Sebastiani 2006)
Bing Liu	Léxico	Libre	(Liu 2012)
Sentiment 140	Léxico	Libre	(Go et al. 2009)
AFINN	Léxico	Libre	(Årup Nielsen 2011)
SenticNet	Léxico	Libre	(Cambria et al. 2014)
VADER	Léxico	Libre	(Borg & Boldt 2020)
CoreNLP	Conjunto de herramientas	Libre	(Manning et al. 2014)
GATE	Kit de herramientas	Libre	(Cunningham 2002)
LingPipe	Kit de herramientas	Libre	(Carpenter 2007)
MALLET	Kit de herramientas	Libre	(McCallum 2002)
OpinionFinder	Conjunto de herramientas	Libre	(Wilson et al. 2005)
LIWC	Conjunto de herramientas	Cuota	(Pennebaker et al. 2001)
AutoML	Conjunto de herramientas	Demo/Cuota	(Zeng & Zhang 2020)
Monkey Learn	Conjunto de herramientas	Demo/Cuota	(Monkey-Learn 2020)
Hootsuite	Herramienta	Demo/Cuota	(HootSuite, WE ARE SOCIAL Y 2020)

3.2.3.3. Metodología para el Análisis de Sentimientos

En la Figura 3.3 se muestra de manera general la metodología que utilizan diversos autores para llevar acabo la implementación de Análisis de Sentimiento, abordando cuatro fases principales.

Figura 3.3: Metodología general de análisis de sentimiento.

1. **Recolección de datos**: Esta fase implica la búsqueda y recopilación de datos, si bien, no existe un estándar para llevar a cabo este proceso, la mayoría de los trabajos publicados toman a *Twitter* como fuente de referencia para obtención de datos, haciendo uso de la *API* de *Twitter* para conectarse y acceder a datos sobre los usuarios de Twitter para su análisis (Kouloumpis et al. 2011, Ray & Chakrabarti 2017).

2. **Limpieza de datos**: La segunda fase consiste en la limpieza y preparación de los datos, en este punto se eliminan aquellas palabras que no contienen mucha información sobre el sentimiento en el texto, comúnmente conocidas como *stopwords*, normalmente suelen estar conformadas por pronombres, preposiciones, adverbios y verbos, así como también se eliminan datos duplicados como *retweets*, páginas *web*, correos electrónicos, *hashtags*, etiquetas de usuarios, caracteres especiales como signos de puntuación, acentos, espacios en blanco y números además de la conversión del texto de mayúsculas a minúsculas.

3. **Detección del sentimiento**: Para la detección del sentimiento existen dos vertientes principales: **i)** enfoque basado en el léxico y **ii)** enfoques basados en aprendizaje supervisado. El primer enfoque es conocido como la aproximación basada en orientación semántica, cada palabra definida en el diccionario está asociada a un valor numérico que define el sentimiento. Por ejemplo *SentiWordNet* es un léxico que asigna tres puntuaciones de sentimiento tales como positividad, negatividad u objetividad a un texto dado (Esuli & Sebastiani 2006)). Por otro lado, se encuentra el léxico de opiniones que determina la polaridad del texto en dos clases (positivos, negativos), este léxico se creó con la recopilación de datos obtenidos desde diferentes sitios de la web creando un léxico informal puesto que normalmente no siempre se siguen las reglas gramaticales en sitios de internet (Liu 2012).

   Un tercer ejemplo enfocado en el léxico sería la herramienta **VADER** *(Valence Aware Dictionary and sEntiment Reasoner)*, basada en reglas y léxico que se adapta a los sentimientos expresados en redes sociales. Usa una combinación de léxico relacionado con el sentimiento, con una serie de palabras que se etiquetan teniendo en cuenta su orientación semántica, que puede ser positiva o negativa (Ichi-Pro 2020-2021).

   Por otro lado, los enfoques supervisados de aprendizaje automático se basan en el análisis manual del texto para la determinación de la polaridad con base en la experiencia del etiquetador, creando así un conjunto de datos que sirven como entrenamiento para la creación de un clasificador de sentimientos automático.

4. **Validación de resultados:** Por último se implementan diversas métricas de validación como la matriz de confusión, esto es para evaluar la precisión de un

modelo de aprendizaje automático, verificando si el clasificador predice de manera correcta los datos.

3.2.3.4. Niveles de Análisis de Sentimientos

Vamos a tener tres niveles distintos para llevar a cabo el análisis de sentimientos de un documento (Sobrino Sande 2018):

- **Análisis a nivel de documento:** Nivel en el cual se va a analizar que tipo de sentimiento se encuentra en un documento pero a través de una visión global como un todo. Se va a clasificar en positivo, negativo y neutro.
- **Análisis a nivel de oración:** Nivel en el que se divide un documento en oraciones individuales y con esto podremos extraer la opinión de cada una de ellas dividiéndolas en opiniones positivas, negativas y neutras.
- **Análisis a nivel de aspecto y entidad:** Nivel que se a analizar con mayor detalle. Consiste en una entidad más pequeña que va a estar formada por distintos elementos, y se expresará una opinión de cada uno de ellos.

Capítulo 4

Estado del Arte

En este capítulo se exponen algunas investigaciones relacionadas con el análisis de sentimiento haciendo uso de diversas librerías y herramientas especializadas en el PLN. En este ámbito la mayoría de los *datasets* se obtienen de *Twitter* ya que es uno de los mayores servicios de información a tiempo real a través de internet, por lo que se podrán obtener grandes cantidades de conjuntos de datos para entrenar nuestros algoritmos y se pueden encontrar *datasets* según temática usando los *hastags*. En la Seccion 4.1 se exponen aquellos trabajos que han utilizado distintas herramientas del PLN para analizar los sentimientos en *Twitter*. En la Sección 4.2, por el contrario, se muestran aquellos trabajos que han utilizado el léxico como técnica y la Sección 4.3 se exponen trabajos que ha utilizado ambos tipos de implementación; primero el léxico y luego el aprendizaje automático. En la Sección 4.4 se van a tratar con trabajos que utilizan el aprendizaje automático para analizar el sentimiento de los usuarios en la aplicación de *Twitter*.

Cabe destacar que gran parte de los trabajos que se realizan en este ámbito contienen un conjunto de datos en inglés, como verá más adelante el idioma en el que está nuestro conjunto de datos es importante a la hora de tomar ciertas decisiones como que tipo de preprocesado o etiquetado se va a utilizar. Por ello, finalmente, en la Sección 4.5 se van a analizar trabajos que utilizan un *dataset* en castellano.

4.1. Técnicas Basadas en Algoritmos que Utilizan Distintas Herramientas

(SAURA et al. 2018) realizan un análisis de sentimientos basado en el *hashtag* *#BlackFriday* usando aprendizaje automático. Su objetivo era medir el sentimiento que tenia el consumidor en *Twitter* en relación con los tres grados de polaridad (positivos, negativos, neutrales) relacionados con dicho evento y las ofertas que se publicaron en la red social usando dicho *hashtag*. Obtuvieron un total de 2,204 *tweets* relacionadas con una serie de empresas. Estas fueron seleccionados según un *ranking* nacional de empresas que se había publicado en "El Economista" en el año 2017 por tener gran prestigio en

las categorias de telefonía, electrónica y tecnología, tales como *Amazon, Xiomi, Asus, Fnac, LG, Media Markt...* entre otros. Este algoritmo fue desarrollado en *Python*, se usó la librería *Monkeylearn* con la que se aplicó el análisis de sentimiento para dividir los *tweets* en los tres grados de polaridad. Como resultado de los experimentos se obtuvo un total de 170 *tweets* negativos en el cual se obtuvo una media de 0.521. Los *tweets* neutrales que se obtuvieron 1327 obtuvieron una media de 0.563. Por último, los *tweets* positivos que fueron 707, tuvieron una media de 0.654. Esta investigación proporciona unos datos que pueden ser utilizados para estrategias de marketing futuras y servirán como un inicio para posteriores investigaciones.

(Martin-Domingo et al. 2019) analizaron opiniones sobre la cuenta de Londres *Heathrow* en *Twitter* desde el punto de vista del análisis de sentimientos para identificar nuevas ideas que pudieran mejorar la calidad del servicio del aeropuerto (ASQ). Desarrollaron un sistema que recopilaba y analizaba los datos. Su diseño incluia una lista de 108 palabras clave que identificaban tweets donde se hablaba de la calidad de dicho ASQ. Su muestra consistió en 4,392 tweets. Su trabajo concluyó que había 23 atributos sobre el ASQ. El análisis de sentimientos se realizó usando las herramientas *Theysay* y *Twinword*. Los resultados mostraron que podían obtener una precisión de 78.7 % y 69.6 % ACC respectivamente, siendo *Theysay* el que obtuvo un mejor rendimiento.

4.2. Técnicas Basadas en el Léxico

(Sabariah et al. 2015) realizaron un análisis de sentimiento basado en el léxico para evaluar el rendimiento que tenía un programa de televisión en Indonesia a través de los comentarios que se hacían en *Twitter* sobre la valoración de dicho programa. El método que se usa es un enfoque basado en el léxico con el que se obtiene la polaridad de los sentimientos, etiquetándolos en positivos y negativos. Este método utiliza la ayuda de un diccionario para clasificar los diferentes *tweets*. Algunos de los pasos que se usan es el manejo de la negación, determinar la polaridad y dar una puntuación a cada entidad del *tweet*. Con todos estos datos se pudo entrenar el clasificador SVM con el que se obtuvo una tasa de precisión del 80 %.

(Cernian et al. 2015) realizaron un análisis de sentimiento basándose en el actual contexto social, tecnológico y económico donde los clientes toman sus decisiones basándose mayoritariamente en la opinión de otros consumidores. Por otro lado, las empresas necesitan una rápida retroalimentación de sus clientes para adaptarse a sus necesidades en tiempo real. La conexión entre estos dos aspectos se basa en herramientas de extracción de opiniones, que procesan automáticamente las reseñas y opiniones de los consumidores sobre productos o servicios. Desarrollaron una arquitectura para extraer y procesar las fuentes de texto utilizando *Stanford POS Tagger* y luego se utilizó el recurso *SentiWordNet*, diseñado para procesar y analizar información sobre sentimientos para el idioma inglés. A cada término se le asigna una puntuación para cada una de las siguientes características:

positividad, negatividad y objetividad. La validación experimental se llevó a cabo en un conjunto de 300 reseñas de productos electrónicos de Amazon que datan de junio de 1995 hasta marzo de 2013. Las reseñas en Amazon se califican con 0 a 5 estrellas. Se demostró una tasa promedio de éxito de la aplicación del 61 %. Para la metodología de validación se comparó los resultados producidos por la aplicación con las calificaciones de estrellas de Amazon, con el fin de obtener una imagen objetiva de la precisión del enfoque. Como desarrollos futuros, se planeó optimizar el algoritmo para que proporcione un procesamiento semántico extendido de fuentes de texto.

4.3. Técnicas Basadas en la Combinación del Léxico y el Aprendizaje Automático

(Nguyen et al. 2018) propusieron un estudio que utilizaba los enfoques de aprendizaje automático supervisado así como técnicas basadas en el léxico. Se creó un script para descargar las reseñas de consumidores de productos de *Amazon*. Etiquetaron como negativas aquellas que tenían de 1 a 3 estrellas de calificación, y como positivas las que tenían 4 y 5. Se eliminaron caracteres especiales de HTML, emojis, palabras vacías y aplicaron procesos de tokenización y lematización. Para ello se utilizó el método de TF - IDF. Se van a comparar un total de 6 algoritmos. Los 3 correspondientes al aprendizaje supervisado son: Logistic Linear Regression (LLR), SVM y aumento de gradiente. Los otros 3 algoritmos que se basan en el léxico son: VADER, *Pattern* y *SentiWordNet*. Los resultados que se obtuvieron demuestran que los clasificadores de aprendizaje automático tienen mejores resultados que los basados en el léxico. Se van a usar las siguientes métricas: matriz de confusión, precisión, sensibilidad y F1. El mejor clasificador es LLR con 90 % de accuracy y VADER el mejor léxico con un 83 %.

(Joyce & Deng 2017) realizaron un análisis de sentimiento en *Twitter* para las elecciones presidenciales de EE. UU. De 2016. Muchas personas expresaron sus gustos o disgustos por un candidato presidencial en particular. Luego se calculó el sentimiento empleando dos enfoques: análisis de sentimiento basado en el léxico utilizando *O*pinionFinder. Este léxico contiene aproximadamente 1.600 y 1.200 palabras positivas y negativas. Se usa un análisis de sentimiento con aprendizaje automático haciendo uso de la herramienta Natural Language Processing Toolkit (NLTK) para implementar el algoritmo Naive Bayes (NB). Los resultados sugieren que *Twitter* se está convirtiendo en la plataforma más confiable en comparación con trabajos anteriores. Al centrarse en los *tweets* 43 días antes de las elecciones (comenzando con el primer debate presidencial), encontramos una correlación de hasta el 94 % con los datos de las encuestas utilizando una técnica de suavizado de promedios móviles. Quizás en el futuro, las encuestas en las redes sociales se incorporen más a los esquemas de votación.

4.4. Técnicas Basadas en el Aprendizaje Automático

(Go et al. 2009) proponen una clasificación de la opinión en *Twitter* mediante supervisión a distancia. Los *tweets* se van a clasificar como positivos y negativos. Esto es útil para los consumidores que desean investigar el sentimiento de los productos antes de comprarlos, o para las empresas que desean monitorear el sentimiento público de sus marcas. No existe una investigación previa sobre la clasificación de la opinión de los mensajes en servicios de *microblogging* como *Twitter*. Los datos de entrenamiento consisten en mensajes de *Twitter* con emoticonos, que se utilizan como etiquetas ruidosas. Este tipo de datos de entrenamiento está disponible en abundancia y se puede obtener a través de medios automatizados. Por ejemplo, :) en un *tweet* indica que el *tweet* contiene un sentimiento positivo y :(indica que el *tweet* contiene un sentimiento negativo. Se comparan los modelos NB, *Maximum Entropy* (MaxEnt) y SVM con los modelos *Unigram, Bigram, Unigram + Bigra*m y Características de *Unigram* + POS. Si se usa *Unigram* el mejor modelo fue SVM con un rendimiento del 82.2 % de *Accuracy*, *Bigram* con NB alcanzan el 81.6 % . Por otro lado, la combinación de *Unigram + Bigram* con MaxEnt tiene un rendimiento de 83.0 % y *Unigram + Tags* POS obtuvo un rendimiento de 81.9 % ACC. Con esto se demuestra que el uso de emojis fue útil para alcanzar una puntuación superior al 80 % de rendimiento.

(Agarwal et al. 2011) construyen modelos para clasificar 2 tareas: una es la tarea binaria de clasificar el sentimiento en positivo y negativo y la otra tarea de tres vías para clasificarlo en sentimiento positivo, negativo y neutro. Experimentamos con tres tipos de modelos: modelo *unigram*, un modelo basado en características y un árbol kernel para analizar los sentimientos en *Twitter*. Para la tarea de clasificación binaria con dos clases de polaridad de sentimiento: positiva y negativa se una un conjunto de datos formado por 1709 instancias para cada clase y, por lo tanto, la probabilidad de referencia es del 50 %. Para la segunda tarea se va a comparar el rendimiento de los tres modelos descritos anteriormente: modelo unigram, modelo basado en características que usa solo 100 características de Senti y el modelo de *kernel* de árbol. Para validar el rendimiento de cada algoritmo, implementaron un modelo SVM en una configuración de validación cruzada. Se observa que los núcleos del árbol superan al unigram y al modelo basado en características en un 2,58 % y un 2,66 % respectivamente. También se experimenta con combinaciones de modelos. La combinación de *unigrams* con modelo de características supera en un 0,78 % a la combinación de *kernels* de árbol con funciones de Senti. Este es el mejor sistema de rendimiento para la tarea positiva frente a la negativa con un 75.39 de *Accuracy*.

(Wan & Gao 2015) analizaron las opiniones de los clientes sobre los servicios de diversas aerolíneas en *Twitter* para la creación de un clasificador de sentimientos. Para ello usaron la API de Twitter donde se obtuvieron datos referentes a las aerolíneas mas famosas de América del Norte y etiquetaron manualmente los datos como positivos, negativos, neutrales. Se utilizón un conjunto de datos de 12864 *tweets* donde se realizó un pre procesamiento de datos eliminando palabras y caracteres que proporcionan poca

información. Se convirtieron los datos en una matriz binaria utilizando N-gram. Por último para la fase experimental utilizaron cuatro clasificadores tales como SVM, NB, Arboles de Decisión C4.5(DTree), Random Forest Random Forest (RF) y un clasificador ensamblado combinando los algoritmos anteriores con una validación cruzada de 10 veces. Como resultados el mejor clasificador individual fue RF con 83.5 % de rendimiento mientras que el clasificador ensamblado mejoró gradualmente la precisión con 84.2 %.

(Buntoro et al. 2016) analizó que *Twitter* no solo se usa para las redes sociales para mantener la amistad, sino que también se usa para promover y hacer campañas. Los usuarios son libres de expresar sus opiniones, incluidas las relacionadas con los candidatos a la presidencia de Indonesia del 2014. Esta investigación se centra en opiniones públicas clasificándola en cinco atributos: muy positiva, positiva, neutral, negativa y muy negativa. Para este proceso de clasificación se uso NB con preprocesamiento de datos usando tokenización, limpieza y filtrado. Se uso un conjunto de datos de 900 tweets y distribuidos a cinco atributos de clase por igual. Este estudio utiliza 5 métodos de tokenización, a saber, unigrama, bigrama, trigrama, 3-5 gramos con un valor mínimo de n = 3 y n = 5 y N-grama. Se obtuvo el mejor resultado cuando el experimento utilizó la combinación de tokenización N-grama, lista de palabras vacías WEKA y emoticonos. Con todo este proceso de investigación, se puede concluir que el análisis de sentimiento con el atributo de cinco clases tiene menor precisión si la comparamos con el atributo de dos o tres clases, con una precisión promedio del 71,9 %. Sin embargo, al utilizar cinco opiniones el sentimiento es más específico y detallado así que la información se presenta con mayor claridad. La tasa de recuperación es de 66,1 % y los valores de TP y TN de 65 %. Además, es necesario desarrollar una lista de palabras irrelevantes y un lematizador indonesio que pueda mejorar la precisión en el análisis de sentimientos.

En (Becerra 2017) se usan 120000 *tweets* usando la etiqueta "#oscars", estos *tweets* están en inglés. Se usa un preprocesado de los *tweets* en bruto donde se hace una tokenización, un *stemming* y más tarde una vectorización usando n-gramas.Después se calcula la frecuencia usando la frecuencia de términos o TF y TF - IDF. Usan validación cruzada de K iteraciones, usan el *accuracy* para evaluar los resultados. Usan 3 clasificadores distintos, máquina de soporte vectorial, árbol de decisión y *Naive Bayes* multinomial usando la librería de *Scikit-Learn* para todos ellos. Para la SVM se obtiene un rendimiento de 87.15 %, con el árbol de decisión se obtiene un 66.85 % usando entropía y con 6 niveles de profundidad como máximo. Por último con *Naive Bayes* multinomial obtienen un 84,6 % usando un factor *alpha* de 0.1.

(Loyola-González et al. 2019), proponen un clasificador basado en patrones de contraste para la detección de *bots* de *Twitter*. Su objetivo era ayudar a expertos a emprender acciones legales contra una cuenta que ha mostrado un comportamiento inusual. En el trabajo exploran el uso de Bayesian Network, KNN, C4.5, NB, RF, SVM, LLR, *Adaptive Boosting, Bagging, PBC4ci, Multilayer Perceptron*, buscaban identificar al mejor clasificador para la detección de *bot*. En su trabajo exponen que el mejor clasificador es

RF con 99 % área bajo la curva (AUC) y coeficiente de correlación (MCC), mientras que el clasificador basado en patrones solo alcanzó 90 % AUC y 91 % MCC.

(Sprogis & Rikters 2020), proponen un *corpus* con *tweets* desde octubre de 2011 hasta abril de 2020. El corpus contiene 2.275.787 *tweets*, de los cuales 155.057 contienen información de medios, 165,335 contienen información de ubicación y 1,297,159 *tweets* mencionan alimentos o bebidas. Se va a comparar una implementación de *Python* del clasificador NB (ejecución cogida de la librería de NLTK) con la implementación de Pinnis del clasificador *Perceptron* (Muischnek & Mü@ ü risep 2018). Se descubrió que la precisión de clasificación más alta (61,23 %) se logra utilizando todos los conjuntos de datos para el entrenamiento.

(Janssens et al. 2015) propone analizar emociones en tiempo real de tweets publicados por diversos usuarios en Twitter. El conjunto de datos va a constar de 7 clases de emociones divididas en: *ira, temor, alegría, amor, tristeza, sorpresa y gratitud*. El corpus va a constar de 249.993 *tweets*, en los que inicialmente se va a realizar un preprocesamiento donde se van a eliminar los *hastags* y se clasifican los mensajes cortos. Se hará uso de una combinación de unigramas, bigramas y tri-gramas junto con TF-IDF. En dicho trabajo se van a comparar 9 clasificadores: Stochastic Gradient Descent (SGD) *modified hubber*,SGD *hinge*,SGD *log*, SGD *perceptron*,LIBLINEAR, MNB, BNB,*Ridge classifier, Nearest centroid*. Se descubrió que el clasificador con mejor accuracy correspondía al SGD *modified hubber* con un 66,64 %.

4.5. Técnicas basadas en el Uso de Dataset en Castellano en el Análisis de Sentimientos en Twitter

El idioma en el que está nuestro conjunto de datos nos va a condicionar a la hora de realizar nuestro modelo ya que es muy diferente tanto la forma de preprocesar el texto como la manera que tenemos de aplicar los clasificadores. Además deberemos usar algunas herramientas extras para conseguir todo esto, como por ejemplo es el uso de un diccionario para poder corregir.

Este problema con el idioma no se hace notar mucho en ya que el preprocesado de este trabajo no es muy preciso debido a que sólo se queda con ciertos tipos de palabras (Holgado et al. 2020). Además, solo usa dos etiquetas, positivo y negativo, por lo que el etiquetado no es muy preciso con el tipo de *dataset* que utiliza. Este trabajo utiliza un conjunto de datos que están relacionados con la ciencia, esto provoca que muchos de los *tweets* que han conseguido estén etiquetados, objetivamente como neutros, sin embargo al tener dos etiquetas solo, se debe poner en alguna de estas dos categorías anteriormente mencionadas. Hay que destacar la gran cantidad de algoritmos que son usados y que se mezclan dichos algoritmos para obtener el mejor resultado. En el trabajo quieren obtener la polaridad de un *dataset* científico mediante el análisis de sentimientos, para ello se usó una muestra de 200.000 *tweets* sobre ciencia en castellano de los cuales 10.000 están etiquetados. EL

primer paso es hacer el preprocesamiento de los textos en bruto, para ello utilizan varios pasos, además cuentan con una primera clasificación de forma manual. En este caso se utilizan dos etiquetas para clasificar los sentimientos de los *tweets*, positivo y negativo. Se utilizan diferentes algoritmos para comparar las distintas métricas(*Recall, Accuracy*, F1 y precision) y ver cual se comporta mejor. Los algoritmos de clasificación que utilizan son: Naive Bayes original, Naive Bayes multinomial, Naive Bayes Bernouilli, Regresión logística, clasificador lineal y Support Vector Classifier (SVC). Se combinan todos estos algoritmos para crear un sistema de votación para elegir el que mejor se comporte teniendo en cuenta la situación, obteniendo una tasa de acierto del 72,32 % con una precisión del 71,5 % , un 77,2 % de Exhaustividad y un valor F del 74,13 %.

En (Hurtado et al. 2015) se contemplan dos tareas diferentes. Esta vez cuentan con un corpus de 7219 *tweets* etiquetados con la polaridad y 60798 *tweets* a los que se les debe asignar polaridad. Cuenta con 6 etiquetas para la primera tarea, N y N+ expresan la intensidad de negatividad, P y P+ que muestran la polaridad positiva en diferentes intensidades, Neutro y NONE que indica la ausencia de polaridad. Para la segunda tarea solo se distinguen 4 etiquetas. Se usan diferentes combinaciones de técnicas de votación. Usando SVM se consiguen las mayores métricas consiguiendo así un *acurracy* de 0.673 cuando existen 6 etiquetas y de 0.725 cuando se usan 4.

(RODRíGUEZ 2018) buscan obtener los resultados electorales de las elecciones de 2017 de Chile basándose en los datos de *Twitter*, intentando obtener la menor diferencia con respecto a los resultados obtenidos realmente. El conjuntos de datos se consigue entre el 5 de Mayo de 2017 y 1 Julio de 2017, los datos son correspondientes al candidato Sebastián Piñera, obteniendo 905 tweets en castellano. Se propone usar dos etiquetas, positivo y negativo para la clasificación de los *tweets*. Se utiliza por un lado TF-IDF con los algoritmos SVM obteniendo un F1 y una exactitud del 90 %, árbol de decisión con el que se logra un 80 % en estas dos medidas mencionadas anteriormente y NB adquiere un F1 del 88 % y una exactitud del 83 %. Además de usar TF-IDF, se utiliza también la clasificación por medio de *Word2Vec* usando los algoritmos SVM para el que se obtiene un F1 del 77 % y una exactitud de 76 %, árboles de decisión con el que se consigue un F1 y una exactitud del 63 % y RF que alcanza como resultado un D1 del 71 % y una exactitud del 70 %. En la Tabla 4.1 se muestra un resumen de los trabajos expuestos y su comparativa en los resultados.

Tabla 4.1: Tabla comparativa

Referencia	Algoritmos	Etiquetas	Resultados
(SAURA et al. 2018)	Monkeylearn	3	
(Martin-Domingo et al. 2019)	Theysay y Twinword	32	P = 78.7 % ACC = 69.6 %
(Sabariah et al. 2015)	Diccionario	2	80 %
(Cernian et al. 2015)	SentiWordNet	3	éxito=61 %
(Nguyen et al. 2018)	LLR, SVM, aumento de gradiente, VADER, Pattern,SentiWordNet	3	LLR=90 % ,VADER= 83 %
(Joyce & Deng 2017)	NB	2	éxito=94 %
(Go et al. 2009)	NB, Maximum Entropy y SVM	2	80 %
(Agarwal et al. 2011)	unigram, basado en características, kernel de árbol y SVM	3	75.39 %
(Wan & Gao 2015)	SVM, NB, Arboles de Decisión C4.5(DTree), RF y un clasificador ensamblado	3	83.5
(Buntoro et al. 2016)	NB	5	P=71,9 %
(Loyola-González et al. 2019)	Bayesian Network, KNN, C4.5, NB, RF, SVM, LLR, Adaptive Boosting, Bagging, PBC4ci, Multilayer Perceptron		ACC=90 %
(Sproģis & Rikters 2020)	Naive Bayes y Perceptrón	3	61,23 %
(Janssens et al. 2015)	SGD*modified hubber*,SGD *hinge*,SGD *log*, SGD *perceptron*,LIBLINEAR, MNB, BNB,*Ridge*, Nearest centroid	2	ACC=66,64 %
(Holgado et al. 2020)	NB original, Naive Bayes multinomial, Naive Bayes Bernouilli, Regresión logística, clasificador lineal y SVC	2	72,32 %
(Hurtado et al. 2015)	SVM	6	ACC = 67.3 %
(RODRíGUEZ 2018)	SVM, arbol de decisión y Naive Bayes	2	80 %
(Becerra 2017)	Máquina de soporte vectorial, árbol de decisión y Naive Bayes multinomial	2	ACC = 87.15 %

Capítulo 5

Metodología de Análisis de Sentimientos Propuesta

En este capitulo se describe la metodología propuesta de análisis de Sentimientos con un *dataset* en castellano, aplicando cada uno de los algoritmos estudiados en capítulos anteriores. En la Sección 5.1 se hace referencia a los pasos llevados a cabo para realizar el preprocesado del corpus. En el se realiza el etiquetado del texto, la eliminación de *stop words* y la corrección ortográfica. Seguidamente, en la Sección 5.2 se describe el modelo que se ha realizado y en la Sección 5.3 los parámetros que se va a utilizar para realizar los experimentos. Finalmente en la Sección 5.4 se muestran los resultados obtenidos.

La metodología propuesta en este trabajo consta de 5 fases que se representan en la Figura 5.1, cada una de ellas se describe en las siguientes secciones.

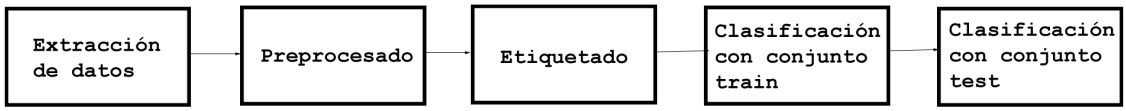


Figura 5.1: Diagrama de la metodología

5.1. Preprocesado del Corpus

El preprocesado se utiliza para normalizar el texto con el fin de dejar preparada la entrada para que los algoritmos de clasificación y análisis de sentimiento puedan procesarlos. En este caso se tienen diferentes niveles de preprocesado para poder usarlos en los diferentes algoritmos. En otros trabajos se opta por clasificar las palabras según su significado semántico, diferenciando así entre tipos de palabras como verbos, adverbios, adjetivos, nombres, etc. y se quedan únicamente con los verbos, adjetivos y sustantivos. En este trabajo se ha preferido no depender de ese clasificado, haciéndolo de forma manual sometiendo los tweets a varios niveles de preprocesado donde los se ha ido limpiando progresivamente de la siguiente manera:

- Eliminación de las menciones, *hashtags links*.
- Eliminación de los caracteres especiales, dejando los signos de puntuación y símbolos numéricos
- Corrección ortográfica y eliminar las *stop words* o palabras de paradas.
- Se pasa a minúsculas y se elimina cualquier símbolo que no pertenezca al alfabeto y eliminación de repeticiones de tweets

Después de cada punto se añade una columna al Dataframe, guardando los resultados para poder usar distintos niveles de limpieza.

5.1.1. Eliminación de las Stop Words

Para este propósito se ha usado el diccionario de *stop words* en español que proporciona NLTK, modificándolo para eliminar las palabras de parada con connotación negativa para no cambiarle el sentimiento al *tweet* negativo. Sin ello, *tweets* como *"no me gusta"* pasaría a ser *'me gusta'*. Como se puede observar en este ejemplo, el sentimiento de la oración cambia notablemente y esto puede hacer que los algoritmos clasifiquen mal los *tweets*. También se han añadido algunas palabras a la lista de *stopwords* que se repetían con gran frecuencia dentro de nuestro corpus científico como pueden ser ciencia o CSIC, que son insignificantes a la hora de valorar el sentimiento de una oración.

5.1.2. Correción Ortográfica

A la hora de tratar los tweets ha sido necesario hacer una corrección ortográfica ya que si no, palabras que están mal escritas podrían confundirse con otras o no ser reconocidas y cambiar totalmente el sentido de una frase. Por tanto, como el corpus está en castellano lo primero que se hace es encontrar alguna forma para realizar esta corrección ortográfica. Se encontraron varios problemas a la hora de hacer esta corrección ya que la mayoría de métodos para corregir lo hacen solo en inglés. Fue todo un reto encontrar una librería que consiguiera hacer bien esta tarea. Al final, después de muchas pruebas, se usó la librería *Hunspell*, ya que era la que mejores resultados nos daba y al estar implementada en C mejoraba su velocidad de forma considerable frente a otros correctores, factor clave ante un corpus de gran tamaño. Esta librería necesita de un diccionario para poder corregir los textos, se consigue un diccionario en formato *.dic* y *.aff* en GitHub[1].

El proceso para corregir fue el siguiente: se mira si la palabra está en el diccionario, si la encuentra devuelve la palabra, si no lo está, le pides a *Hunspell* sugerencias para esa palabra, si existe alguna sugerencia se coge la mejor, (*Hunspell* te la pone en primera posición en la lista de sugerencias), si no hay ninguna sugerencia para esa palabra se devuelve la palabra de partida.

[1] https://github.com/titoBouzout/Dictionaries

Se relatan aquí todos los intentos que se realizaron antes de llegar a *Hunspell*. Se intentó hacer la corrección con un código en lugar de usar librerías, esto daba muchos problemas ya que tardaba mucho, era muchísimo más ineficiente, por lo que se planteó la opción de probar librerías ya que estaban desarrolladas por debajo en C y eran más rápidas.

La primera librería que se probó fue *SPellChecker*, esta necesitaba de un diccionario en formato .txt, para ello se usó un libro escrito en castellano ya que pareció la opción más sencilla para empezar. *SpellChecker* busca la palabra dentro de su diccionario, si no la encuentra tiene dos opciones, o te sustituye la palabra por una parecida o te la deja como está. El principal problema que se encontró con *SpellChecker* es que al usar un libro como diccionario, este cuenta con un vocabulario bastante limitado, por lo que muchas palabras no las encuentra y acaba sustituyendolas por otras muy diferentes que le quitan sentido a la frase original.

Luego se decidió probar con *TextBlob*. Se trata de una librería de procesamiento de texto para *Python*. Esta librería da problemas a la hora de usar el diccionario en castellano, ya que no es tan completo como el de inglés.

Se tuvo un problema para llegar a la corrección ortográfica que parecía más óptima, esto sucedió ya que el corpus estaba en castellano, problema se repitió a lo largo de todo el proyecto.

5.1.3. Etiquetado

Se ha podido observar que en otros trabajos se tiene un etiquetado binario de *positivo* o *negativo*, con ésto, se decidió ampliarlo e introducir una tercera opción, *neutro*. Esto obligó a re-etiquetar todo el corpus y ,debido a su tamaño, se decidió hacer mediante un etiquetador automático, comparando *VADER* y *TextBlob*, donde se escogió el mejor de los dos. Estos etiquetadores requieren de un texto en bruto o poco limpio, por lo que se decide pasarle los *tweets* con un ligero nivel de preprocesado.

El primer problema encontrado, como ya se ha repetido en otras ocasiones, se basa en que los etiquetadores funcionan únicamente con textos en inglés, así que hizo falta traducirlos antes de pasarlos por el etiquetado. Para ello se usó una librería de *python* llamada *translators* que contiene varios traductores, tras investigar se prueba con dos de ellos, el traductor de *Google* y el traductor de *Bing*, a la hora de hacer el etiquetado y comparar resultados. Debido a la necesaria traducción del corpus, se advierte que se pierde exactitud a la hora de etiquetarlo.

El siguiente problema a afrontar fue que el servidor del traductor no estaba preparado para recibir tantas peticiones de traducción de un mismo individuo en tan poco tiempo, por lo que aceptaba un máximo de unas 300 peticiones seguidas antes de bloquearse. Para solucionar este problema se probaron varios enfoques.

1. Se añadió un pequeño *delay* entre petición y petición para no saturar al servidor, pero no funcionó.

2. También se probó a hacer de forma manual, pero debido al gran tamaño del corpus era inviable y más teniendo en cuenta que esta traducción se tenía que realizar dos veces, uno por cada traductor.

3. También se intentó crear varios hilos de ejecución esperando que creara diferentes conexiones con el servidor pero no fue así.

4. Por último, se consiguió encontrar una fórmula que consistía en ejecutar las peticiones en rangos de 300 *tweets* y clasificarlos. Una vez hecho esto, se procedió a ejecutar un comando de consola para terminar este programa y lanzarlo de nuevo, creando así una conexión completamente nueva con el servidor y solventando el problema.

Pese a esto, hubo algunos casos de problemas esporádicos como reconexión con el servidor durante una petición, lo cual provocaba un error durante su etiquetado. Aunque esporádico, cerca del 1 %, debido al tamaño del corpus la suma de los fallos era ligeramente considerable. Así que se hizo una segunda pasada de esos *tweets*, a los que se asignó un valor de fallo durante el anterior intento, hasta obtener el etiquetado correcto. El tiempo estimado para hacer esta traducción junto con el etiquetado para las dos librerías fue de 13h. Tiempo considerable a la vez aceptable ya que solo se tenía que realizar una única vez.

El mecanismo usado para clasificar los *tweets* fue el mismo para los dos tipos de etiquetadores, el resultado de estos era un número entre -1 y 1, clasificando así como *positivos* los que tenían una puntuación entre (0.05, 1], los *neutros* entre [-0.05, 0.05] y los *negativos* contaban con una puntuación entre [-1, -0.05). Dentro de los neutros se decide separarlos en *neutro-negativo*, *neutro* y *neutro-positivo* para comprobar que estuvieran equilibrados y no hubiera una inclinación hacia positivos o negativos dentro de los mismos.

Además se hizo una diferenciación en las estadísticas y comprobar que el funcionamiento de los etiquetadores era el adecuado y si estaba equilibrado, entre tres tipos de neutro (Neutro, neutro positivo y neutro negativo):

- El neutro negativo tiene puntuaciones [-0.05, 0).
- El neutro positivo entre (0, 0.05].
- la puntuación del neutro es 0.

Si clasifica muchos más tweets como uno de los dos neutros no puros que el otro o como el neutro puro, indicaría que está desequilibrado. Los resultados obtenidos se presentan en las Tablas 5.1 a 5.4.

Tabla 5.1: VADER con traductor de Google

Vader / Corpus	negativo [-1, -0.05)	neutro-neg [-0.05, 0)	neutro 0	neutro-pos (0, 0.05]	positivo (0.05, 1]	Total
negativo	1993	34	1379	39	1156	4601
positivo	701	14	2485	35	1640	4875
Total	2694	48	3864	74	2796	9476

Tabla 5.2: VADER con traductor de Bing

Vader / Corpus	negativo [-1, -0.05)	neutro-neg [-0.05, 0)	neutro 0	neutro-pos (0, 0.05]	positivo (0.05, 1]	Total
negativo	2116	31	1270	35	1149	4601
positivo	707	13	2464	39	1652	4875
Total	2823	44	3734	74	2801	9476

Tabla 5.3: Textblob con traductor de Google

TextBlob / Corpus	negativo [-1, -0.05)	neutro-neg [-0.05, 0)	neutro 0	neutro-pos (0, 0.05]	positivo (0.05, 1]	Total
negativo	935	106	2303	118	1139	4601
positivo	452	76	2544	105	1698	4875
Total	1387	182	4847	223	2837	9476

Tabla 5.4: Textblob con traductor de Bing

TextBlob / Corpus	negativo [-1, -0.05)	neutro-neg [-0.05, 0)	neutro 0	neutro-pos (0, 0.05]	positivo (0.05, 1]	Total
negativo	1010	113	2158	127	1193	4601
positivo	470	83	2510	105	1707	4875
Total	1480	196	4668	232	2900	9476

Los resultados de los distintos etiquetadores con las diferentes traducciones se resumen en la Tabla 5.5. Se concluye que el mejor etiquetado es el de VADER con el traductor de *Bing*, ya que nos ofreció los mejores etiquetados comparando con el original y el menor número de etiquetado como neutro, manteniendo una proporción similar entre positivos, negativos y neutros, con un ligero desfase de los neutros. Esto, nos permitió saber que nuestro etiquetado estaba equilibrado y que por lo tanto es el mejor para entrenar nuestros algoritmos.

Tabla 5.5: Comparativa resultados Vader y TexBlob

Algoritmos / Resultados	VADER		TextBlob	
	Google	Bing	Google	Bing
Porcentaje de aciertos	62,33 %	67 %	62.33 %	62,03 %
Porcentaje de neutros	55,42 %	39,4 %	55,42 %	53,78 %

5.2. Descripción del Modelo

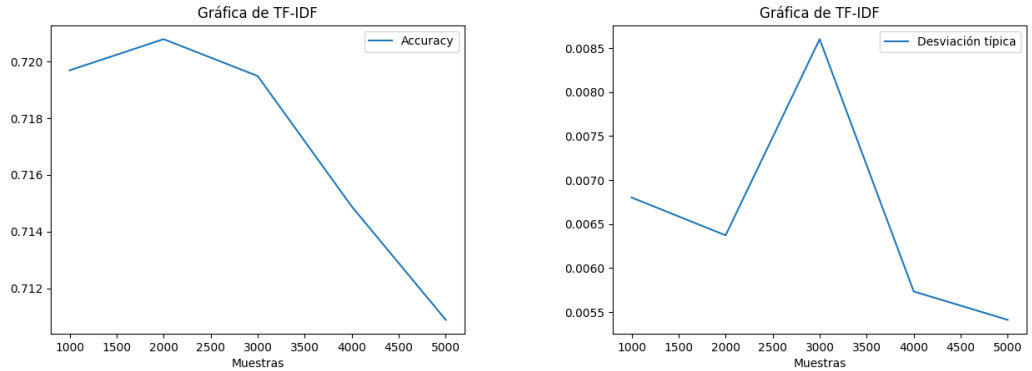
Después del **etiquetado**, y teniendo un archivo *.csv* con el que se pueda trabajar a la hora de probar distintos clasificadores, se procedió a hacer una extracción de los datos.

5.2.1. Paso de los Datos a Frecuencias TF-IDF

Primero de todo, se determinaron cuales eran las palabras más relevantes a la hora de clasificar *Tweets* en **positivos, negativos y neutros**, y qué cantidad de ellas sería la que daría el mejor rendimiento.

Para ello, se utilizó el *algoritmo* Chi^2 para extraer del *corpus* un vocabulario cuyas n palabras eran las más relevantes a la hora de clasificar un documento en positivo, negativo o neutro. Los valores de n que se probaron fueron 1000, 2000, 3000, 4000 y 5000 palabras. Liu & Setiono (1995)

Con los respectivos vocabularios, se ha creado un *TFIDFVectorizer*, el cual transformó los datos y exportó una matriz TF-IDF, para indicar la importancia de cada palabra en cada documento. A partir de esa matriz TF-IDF, se probó cómo de bueno había sido el rendimiento de ese vocabulario, haciendo uso de un **clasificador**. En este caso, se tomó como clasificador el LLR sin parámetros modificados, tratando en todo momento que fuera un clasificador lo más simple posible. Se considera que si a un clasificador simple se le entrena con una matriz TF-IDF, y da buen rendimiento, otros clasificadores más complejos también funcionarán bien con dicha matriz. Como muestra la Figura 5.2, la configuración que mejor resultado da usar las 2000 mejores palabras.



(a) Tasa de acierto

(b) Desviación típica

Figura 5.2: Gráficas usando TF/IDF

5.2.2. Entrenamiento y Comparación de Clasificadores

Una vez extraída la partición de datos de entrenamiento y la partición de datos de test, se procedió a probar el rendimiento de diversos clasificadores. *GridSearchCV* es

una librería de *SKLearn* que permite probar clasificadores con todas las configuraciones de hiperparámetros posibles que se le introduzcan. Por ejemplo, Se supone que a *GridSearchCV* se le introdujera el clasificador *SGD*, el cual tiene los hiperparámetros *max-iter* y *loss*. Si se ponen como valores posibles *max-iter = 10, 100* y *loss = hinge, squaredhinge*. Se tendrían 4 configuraciones *(2 x 2)*. El código está estructurado en una clase abstracta *Classifier*, de la cual heredan todos los clasificadores. Dicha clase requiere obligatoriamente de:

- Un atributo *Grid-parameters* que contiene los valores para los hiperparámetros que se introducirán en *GridSearchCV*.
- Una función *clasify()*, la cual se encarga de pasarle el clasificador a la clase abstracta para que haga un Grid y pruebe todas las configuraciones deseadas.

Los clasificadores fueron probados uno a uno con todas las configuraciones posibles de los valores introducidos en *Grid-Parameters*.

A cada una de esas configuraciones se le aplicó la validación cruzada en 5 *folds*.

5.2.3. Predicción

Por último, una vez se supieron los resultados de los experimentos, y cuales eran las mejores configuraciones para cada clasificador, se pasó a comprobarlos con la muestra de test. Se guardaron en archivos *pickle* los mejores clasificadores, entrenados con la muestra de entrenamiento, y se predijo la muestra de *test*. El resultado de dicha predicción, se guardó para cada configuración del clasificador en una matriz de confusión. En la Figura 5.3 se puede ver una representación visual del modelo.

Según los experimentos de la Sección 5.4 se eligió como mejor algoritmo el SGD. Esto se debe a que nos calculaba bastante bien la precisión y el *recall*, estando muy balanceadas con lo que nos clasificaba bien las clases. Sin embargo también se podía haber elegido el algoritmo XGBoost ya que no hay una gran diferencia en el *accuracy* y también clasificaba bien.

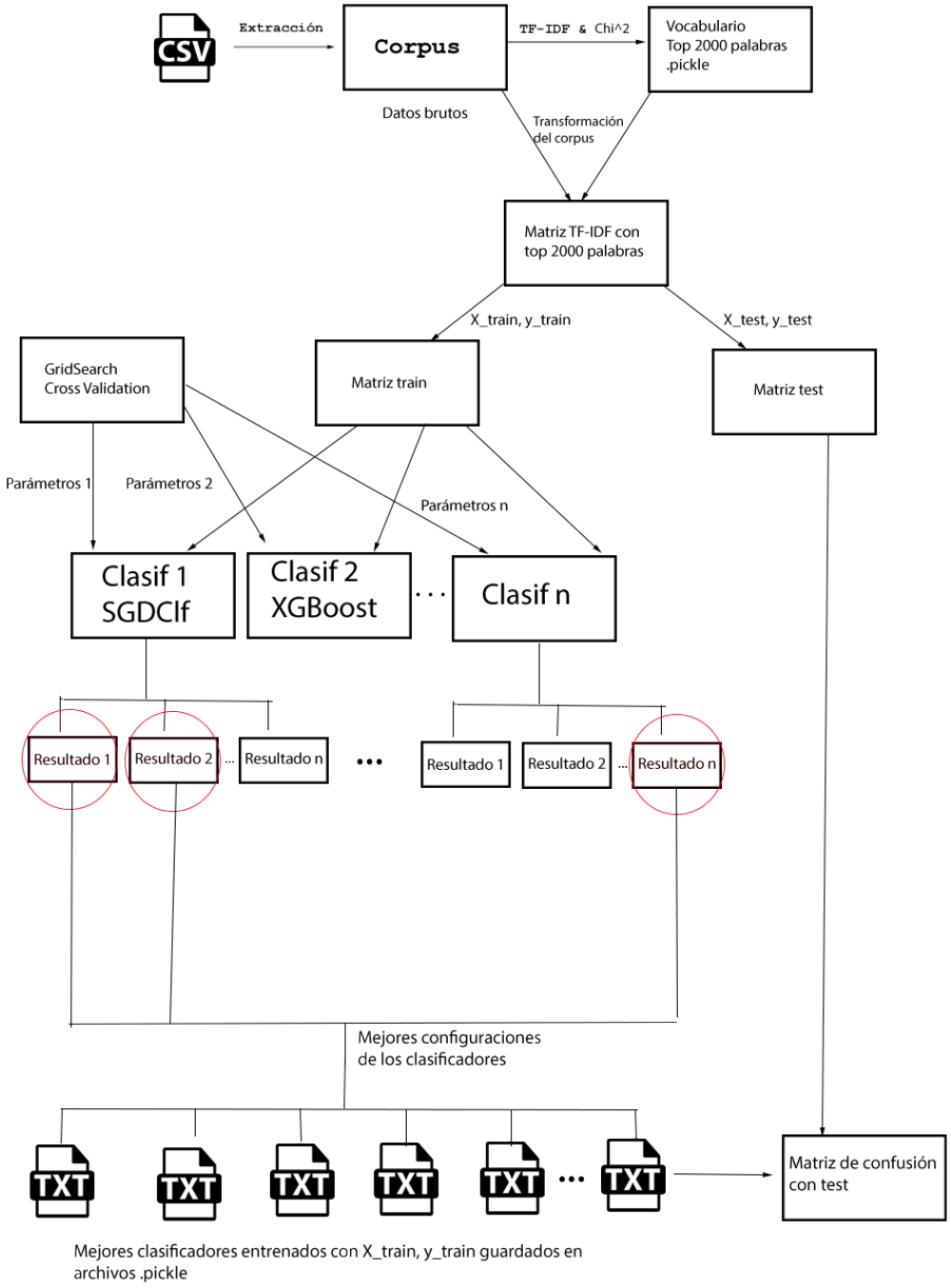


Figura 5.3: Diagrama del modelo

5.3. Parametrización de los Algoritmos

A continuación se detallan los parámetros utilizados en cada uno de los algoritmos con los que se va a realizar los experimentos.

5.3.1. Descenso de Gradiente Estocástico

Para el caso concreto del SGD, se tomaron como variables principales el número de iteraciones máximas, y la función de pérdida:

- El **número de iteraciones máximas** (**max-iter** en el código) es un parámetro que se toma para evitar que el clasificador ajuste demasiado los *biases* y los *pesos*, corriendo así el riesgo de sobreaprender. Se han tomado los valores para iteraciones máximas: *(1,2,5,10,20,50,100,200,500 y 1000)*

- Por otro lado, la función de pérdida o función de coste, es aquella que se trata de minimizar, a través del *descenso del gradiente estocástico*, y que se toma de referencia, para saber la correlación entre los parámetros de la red neuronal y su rendimiento. Se tomaron como funciones de pérdida, las siguientes: *(Hinge loss, Log Loss, Huber Loss modificada, Hinge Loss elevada al cuadrado y Perceptrón)*

Siempre se priorizará la consistencia de los resultados a la hora de hacer validación cruzada (la menor desviación típica) antes que una buena media.

5.3.2. Aumento de Gradiente Extremo

Los parámetros principales que se han tomado para este algoritmo han sido: *booster, n_estimators y learning_state*. Sin duda el parámetro que más relevancia toma es **booster**.

- ***booster*** Se utilizaron los 3 valores posibles para este parámetro.

- ***n_estimators*** En estos experimentos se han usado: (50, 100, 200, 300)

5.3.3. Los K Vecinos más Cercanos

Para el caso de KNN, se han tomado como variables principales *metric*, *n_neighbors* y *weights*:

- ***n_neighbors*:** se refiere al número de vecinos que se van a tener en cuenta, por defecto toma el valor 5. Se le da como opciones (2, 3, 5).

- ***metric*:** es como mide la distancia que va a tomar con los vecinos, por defecto toma *minkowski*. Se utilizan (*minkowski, euclidean, manhattan*)

- ***weigths*:** se refiere al peso que va a tomar en la prediccion, por defecto toma el valor *uniform* que le da el mismo peso en todo el "vecindario"pero también puede tomar el valor *distance*, que le otorga la inversa de la distancia.

5.3.4. Árboles de Clasificación

Para este algoritmo se han usado como principales parámetros:

- ***criterion*:** Se le asignan los dos valores que puede tomar este parámetro gini y entropy.
- ***max_depth*:** Los valores usados para para esto son *(None, 2, 3, 5, 10, 12)*.
- ***min_samples_split*:** Como valores utilizados están *(2, 5, 10, 100)*.
- ***min_samples_leaf*:** Se han usado los valores *(2, 5, 10)*

5.3.5. Bosques Aleatorios

Para este algoritmo se han usado como principales parámetros:

- ***criterion:*** Se le asignan los dos valores que puede tomar este parámetro gini y entropy.
- ***max_depth:*** Los valores usados para para esto son *(None, 2, 3, 5, 10, 12)*.
- ***min_samples_split:*** Como valores utilizados están *(2, 5, 10, 100)*.
- ***min_samples_leaf*:** Se han usado los valores *(2, 5, 10)*
- ***n_estimators*:** Con los valores siguientes *(50, 75, 100, 150)*

5.3.6. Clasificador Bayesiano Ingenuo Multinomial

El Clasificador Bayesiano Ingenuo Multinomial se usa para la clasificación de elementos que tengan características discretas (por ejemplo, contar el número de palabras que permitan clasificar un texto).

Este algoritmo implementa el Clasificador Bayesiano Ingenuo para datos distribuidos multinomialmente.

El parámetro principal que se ha utilizado en este algoritmo ha sido *alpha*.

- ***alpha***: Indica la prioridad de suavizado. Si $\alpha >= 0$ se tiene en cuenta las características que no están presentes en las muestras de aprendizaje y evita una probabilidad de cero en cálculos que se hagan posteriormente. Si $\alpha = 1$ se llama suavizado de Laplace, mientras que si $\alpha < 1$ es el suavizado de Lidstone.
- ***fit_prior***: Parámetro que aprende o no las prioridades previas de la clase. si es falso se utilizará una prioridad uniforme.

Cuanto mas bajo sea el valor de *alpha* mejor rendimiento obtiene el algoritmo, siempre y cuando la prioridad previa sea positiva.

5.3.7. Máquina de Soporte Vectorial

LinearSVC permite ajustar modelos SVM con kernel lineal. Es similar a SVC cuando el parámetro kernel='linear' , pero utiliza un algoritmo más rápido.

A pesar del gran número de parametros que tiene el algoritmo los principales que se han usado son: *penalty*, *loss*, *C*, *max_iter*

- ***penalty***: Parámetro que especifica el tipo de penalización. El éstandar de SVC es l2, tal como se ha comprobado en los experimentos.
- ***loss***: Especifica la función de pérdida. Hinge o también llamada *bisagra* va a ser el estándar de SVM, mientras que *squaped_hinge* va a ser el cuadrado de hinge.
- **C**: Es el parámetro que se encarga de la regularización. La fuerza de la regularización es inversamente proporcional a C. Básicamente, este parámetro le dice al modelo cuánto desea evitar equivocarse.
- max_iter: Parámetro que marca el número máximo de iteraciones que se ejecutarán.

Tras realizar los experimentos se comprobaron que aun variando el máximo de iteraciones siempre se obtenían los mismos resultados. El parámetro que demuestra el cambio en la media y la desviación típica es C.

5.3.8. Clasificador Bayesiano Ingenuo de Bernoulli

Para el el Clasificador Bayesiano Ingenuo de Bernoulli se han tomado como variables ***binarize***, ***alpha*** y ***fit-prior***. Tras los experimentos se observa que la variable más relevante es el *binarize*, trabajando bien con valores cercanos a 0.1. No ha habido una gran diferencia al variar los valores de *alpha* o *fit-prior*, observando un empeoramiento con valores de *alpha* cercanos a 0.

5.3.9. Regresión Logística

La regresión logística se han utilizado los siguientes parámetros:

- **C**: Es el parámetro que se encarga de la regularización. La fuerza de la regularización es inversamente proporcional a C. Básicamente, este parámetro le dice al modelo cuánto desea evitar equivocarse.
- ***max_iter***: Parámetro que marca el número máximo de iteraciones que se ejecutarán.

5.4. Experimentos

A continuación se van a detallar los experimentos realizados para distintos algoritmos. En ellos se han usado las siguientes métricas: *accuracy* para toda la matriz de confusión, y *recall* y precisión para las clases concretas; teniendo en cuenta que un buen modelo es aquel cuyo rendimiento en *accuracy* permanece estable a la hora de probarlo con la muestra de *train* y la de *test* y tanto su *recall* como precisión tienen un valor elevado. La relación entre precisión y *recall* es la siguiente (Arce 2019*b*):

- Alta precisión y alto *recall*: El modelo consigue separar las clases de forma precisa.
- Baja precisión y alto *recall*: El modelo no detecta la clase de forma precisa, pero cuando lo hace es altamente fiable.
- Alta precisión y bajo *recall*: Cuando el modelo decide que una muestra pertenece a una cierta clase, en general acertará. Pero habrá muestras de dicha clase, que no llegará a recuperar.
- Baja precisión y bajo *recall*: El modelo no logra separar la clase de forma confiable.

Dado que han salido diversos resultados, a través de todas las combinaciones de parámetros que se han podido hacer, se han analizado aquellos con mejor rendimiento.

5.4.1. Evaluación del Algoritmo Descenso de Gradiente Estocástico

En la Tabla 5.6 se puede observar un alto *accuracy* en las configuraciones elegidas para el subconjunto de *train*. No presenta una alta desviación típica por lo que se puede deducir que no llega a sobreaprender. Se puede comprobar un ligero estancamiento a partir de 20 iteraciones, llegándose a estancar en 50, por lo que se puede concluir que el mejor mejor resultado se obtiene con 20 iteraciones. Se concluye que 20 es el mejor resultado ya que, si bien no es el más alto numéricamente, si se tienen en cuenta el resto de resultados, como la desviación típica o el tiempo, se puede observar un salto considerable, para la ínfima diferencia del *accuracy* obtenido. Se obtiene un tiempo promedio de 0.0342 segundos.

Tabla 5.6: Mejores 5 configuraciones del modelo con el algoritmo SGD

Num. Iteraciones	Función pérdida	Media	Desviación típica	Tiempo
20	Hinge	0.737	0.015	0.03
50	Huber Modificada	0.746	0.019	0.057
100	Huber Modificada	0.745	0.017	0.057
10	$Hinge^2$	0.696	0.003	0.016
5	Perceptron	0.695	0.009	0.011

Con los modelos alcanzados de las anteriores configuraciones se ha obtenido las matrices de confusión ante el subconjunto de *test*, eligiendo únicamente las de mejor resultado; siendo en este caso las configuraciones 2 y 3.

Tabla 5.7: Matriz de confusión con los resultados del test con el algoritmo SGD

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	612	166	95
Neutro	77	1071	88
Positivo	83	212	593

Teniendo en cuenta la matriz de confusión de la Figura 5.7 se han calculado las métricas de la Tabla 5.8.

Tabla 5.8: Resultados de las métricas para el *test* con el algoritmo SGD

Métrica	Negativo	Neutro	Positivo	Media
Precisión	79,27 %	73,91 %	76,41 %	76,53 %
Recall	70,10 %	86,65 %	66,78 %	74,51 %
Accuracy	75,94 %			

Analizando estos resultados se puede concluir que el *accuracy* se ha mantenido tanto en *test* como en *train*. Por otro lado, se pueden separar las 3 clases de forma rigurosa teniendo en cuenta la relación *precisión-recall*.

Comparando estos resultados con el trabajo (Janssens et al. 2015) se observa una correlación entre los resultados obtenidos en ambos casos. Se puede ver como en dicho trabajo, en el cual se usan 7 clases, se logra alcanzar un *accuracy* en torno al 66 % de media, mientras que en este trabajo se han obtenido unas clasificaciones en torno al 73 % debido al inferior número de clases a predecir.

5.4.2. Evaluación del Algoritmo Aumento del Gradiente Extremo

En la Tabla 5.9 se puede observar un alto *accuracy* con las configuraciones elegidas para el *train*. Presenta una desviación típica ligeramente superior a lo deseado aunque no tan alto como para considerar que el modelo ha llegado a sobreaprender.

El tiempo promedio en la ejecución del algoritmo ha sido 0.0118 segundos.

Tabla 5.9: Mejores configuraciones del modelo con el algoritmo Aumento del Gradiente Extremo

n_estimators	Media	Desviación típica	Tiempo
50	0.745	0.024	0.013
100	0.745	0.025	0.012
200	0.745	0.024	0.021
300	0.746	0.025	0.013

Con los modelos alcanzados de las anteriores configuraciones se ha obtenido las matrices de confusión, Tabla 5.10, ante el subconjunto de *test*, eligiendo únicamente las de mejor resultado; siendo en este caso la configuración 2.

Tabla 5.10: Matriz de confusión del test con el algoritmo Aumento del Gradiente Extremo

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	633	160	80
Neutro	96	1027	113
Positivo	86	232	570

Tras analizar los resultados obtenidos con el *test* (Tabla 5.11), se puede observar como el modelo ha generalizado rigurosamente manteniendo un *accuracy* estable acorde al los resultados del *train*. Examinando la precisión entre las diferentes clases, se puede decir que el modelo clasifica bien, aunque no llega a recuperar tantas muestras positivas como en el resto debido a su bajo *recall*.

Tabla 5.11: Resultados de las métricas para test con el algoritmo Aumento del Gradiente Extremo

Métrica	Negativo	Neutro	Positivo	Media
Precisión	77,67 %	72,37 %	74,70 %	74,92 %
Recall	72,51 %	83,09 %	64,19 %	73,26 %
Accuracy	74,41 %			

5.4.3. Evaluación del Algoritmo K Vecinos más Cercanos

Con los modelos alcanzados de las anteriores configuraciones se ha obtenido las matrices de confusión, Tabla 5.12 ante el subconjunto de *test*, eligiendo únicamente las de mejor resultado; siendo en este caso la configuración 4.

Tabla 5.12: Matriz de confusión con los resultados del test con el algoritmo K Vecinos más Cercanos

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	621	182	70
Neutro	212	888	136
Positivo	262	263	363

Si se comparan los resultados de *accuracy* obtenidos en test (Tabla 5.13) y en *train* (Tabla 5.14), se puede ver que están en torno al mismo valor, eso quiere decir que no ha sobreaprendido a pesar de ser este, un algoritmo que tiende a ello. Lo que si se puede observar es la gran diferencia que existe entre el *recall* de las clases negativa y neutra frente a la clase positiva, el modelo no está llegando a recuperar una gran cantidad de muestras positivas. Se puede comparar con el trabajo (Taneja et al. 2014) donde se obtienen unos resultados similares.

Tabla 5.13: Resultados de las métricas para el test con el algoritmo K Vecinos más Cercanos

Métrica	Negativo	Neutro	Positivo	Media
Precisión	56,71 %	66,62 %	63,80 %	62,37 %
Recall	71,13 %	71,84 %	40,88 %	61,28 %
Accuracy	62,46 %			

Tabla 5.14: Mejores 5 configuraciones del modelo con el algoritmo K Vecinos más Cercanos

metric	n_neighbors	weigths	Media	Desviación típica	Tiempo
minkowski	2	distance	0.614	0.026	0.003
minkowski	3	uniform	0.587	0.026	0.003
minkowski	3	distance	0.622	0.024	0.003
euclidean	2	distance	0.614	0.026	0.004
euclidean	3	distance	0.622	0.024	0.004

5.4.4. Evaluación del Algoritmos de Árboles de Decisión

Se puede observar en la Tabla 5.15 que da claramente peores resultados que otros clasificadores, y que incluso alterando hiperparámetros tales como la profundidad o aquellos relacionados con la generación de hojas, no consigue mejora. Si se compara con (Becerra 2017) se puede comprobar que estos resultados son completamente normales ya que el peor clasificador que se usa en este trabajo también son los árboles de clasificación, aunque en este obtienen un resultado 10 puntos por encima esto se podría deber a que solo se usan dos clases. El tiempo promedio de las configuraciones del árbol de clasificación, ha sido 0.1106 segundos.

Tabla 5.15: Mejores 5 configuraciones del modelo con el algoritmo árboles de decisión

criterion	max_depth	min_samples_leaf	min_samples_split	Media	Desviación típica	Tiempo
entropy	12	2	100	0.494	0.025	0.081
entropy	None	10	100	0.501	0.030	0.128
gini	None	10	100	0.496	0.019	0.093
gini	None	5	100	0.502	0.019	0.111
gini	None	2	100	0.509	0.033	0.140

Se puede examinar en la Tabla 5.15 que si las configuraciones elegidas no llegan a dar malos resultados en el *accuracy*, tampoco dan unos resultados comparables a otros algoritmos pero mantiene una varianza muy parecida, lo que indica que ha podido sobreaprender. Con los modelos obtenidos de las anteriores configuraciones se ha realizado las matrices de confusión ante el subconjunto de *test*, eligiendo únicamente las de mejor resultado; siendo en este caso la configuración 3.

Una vez obtenida la matriz de confusión (Tabla 5.16) se proceda a calcular las métricas que nos permitiran examinar los resultados alcanzados.

Tabla 5.16: Matriz de confusión del test con el algoritmo árboles de decisión

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	373	284	216
Neutro	193	815	228
Positivo	244	348	296

En la Tabla 5.17 se puede examinar que el *accuracy* obtenido en el conjunto de test es muy similar al del conjunto de *train*, esto es una buena señal ya que se podría afirmar que aunque árboles de decisión es un algoritmo que tiende a sobreaprender, en este caso no lo ha hecho. Aún así se obtiene una baja precisión y un bajo *recall* para la clase positiva, lo que significa que el modelo no consigue distinguirla correctamente.

Tabla 5.17: Resultados de las métricas para test con el algoritmo árboles de decisión

Métrica	Negativo	Neutro	Positivo	Media
Precisión	46,05 %	56,32 %	40.00 %	47,46 %
Recall	42,73 %	65,94 %	33,33 %	47,33 %
Accuracy	49.52			

5.4.5. Evaluación del Algoritmo Bosques Aleatorios

Al ser Bosques Aleatorios un algoritmo mejorado de árboles de decisión, se esperaba que mejorase los resultados de estos, se puede observar que es así (Tabla 5.18), llegando a dar resultados 10 puntos por encima, aún así no se han llegado a obtener los resultados de algoritmos más sofisticados que llegaban a rozar el 75 %. Por otra parte, el tiempo promedio ha sido de 2.2946 segundos, siendo este el más alto de todos los clasificadores.

Tabla 5.18: Mejores 5 configuraciones del modelo con el algoritmo Bosques Aleatorios

criterion	min_samples_leaf	min_samples_split	n_estimators	Media	Desviación típica	Tiempo
gini	1	50	100	0.612	0.033	2.200
gini	1	50	150	0.612	0.030	3.311
gini	2	5	75	0.610	0.030	1.232
entropy	2	5	100	0.610	0.027	1.898
entropy	2	5	150	0.610	0.029	2.832

Con los modelos alcanzados de las anteriores configuraciones se ha obtenido las matrices de confusión, Tabla 5.19, ante el subconjunto de *test*, eligiendo únicamente las de mejor resultado, en este caso se ha considerado la cuarta como mejor.

Tabla 5.19: Matriz de confusión del test con el algoritmo Bosques Aleatorios

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	509	292	72
Neutro	107	1063	66
Positivo	141	413	334

Una vez más, se puede examinar en la Tabla 5.20 que la clasificación de la clase positiva deja mucho que desear, obtiendo un alto nivel de precisión pero un *recall* muy por debajo de la media. Como se puede observar en (da Silva et al. 2014) los resultados obtenidos se puede decir que son parecidos a los que aquí se presentan, en algunos casos llegan a superarlos en 10 puntos pero esto se podría deber al uso de bolsas de palabras.

Tabla 5.20: Resultados de las métricas del test con el algoritmo Bosques Aleatorios

Métrica	Negativo	Neutro	Positivo	Media
Precisión	67,24 %	60,12 %	70,76 %	66,04 %
Recall	58,30 %	86 %	37,61 %	60,64 %
Accuracy	63,59 %			

5.4.6. Evaluación del Algoritmo Bayesiano Ingenuo Multinomial

Los resultados obtenidos con los mejores parámetros han alcanzado un buen desempeño con valores de *alpha* cercanos a 0.01, todo esto se puede observar en la tabla 5.21. El tiempo promedio de ejecución de el algoritmo ha sido 0.0118 segundos.

Tabla 5.21: Configuraciones del modelo con el algoritmo Multinomial

alpha	fit_prior	Media	Desviación típica	Tiempo
0.001	True	0.756	0.032	0.010
	False	0.739	0.018	0.009
0.01	True	0.754	0.029	0.008
	False	0.738	0.018	0.010
0.1	True	0.749	0.026	0.012
	False	0.737	0.020	0.010

Tras hacer el *test* de estos modelos, la configuración 4 es la que ha obtenido mejores resultados. Se han realizado la matriz de confusión (Tabla 5.22) y a partir de esta se realiza el estudio de las métricas (Tabla 5.23). Comparándolo con los resultados que obtuvieron en (Holgado et al. 2020) se considera que se han mejorado ligeramente sus resultados aún teniendo una clase más. Sin embargo, al compararlos con (Becerra 2017) se observa que su modelo tiene un mejor desempeño con una configuración similar, esto podría deberse a que se centran en un tema que tiende a estar más polarizado.

Tabla 5.22: Matriz de confusión con los resultados del test con el algoritmo Multinomial

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	679	86	108
Neutro	166	921	149
Positivo	99	161	628

Tabla 5.23: Resultados de las métricas con el algoritmo Multinomial

Métrica	Negativo	Neutro	Positivo	Media
Precisión	71,93 %	78,85 %	70,96 %	73,91 %
Recall	77,78 %	74,51 %	70,72 %	74,34 %
Accuracy	74,34 %			

5.4.7. Evaluación del Algoritmo Regresión Logística

Los resultados de la Tabla 5.24 muestran las mejores configuraciones del modelo, en el cual puede observarse un estancamiento a partir de 50-100 iteraciones, obteniendo un *accuracy* entorno al 70 %, unos resultados bastante prometedores. Por otro lado, el tiempo medio de ejecución ha sido 0.0324 segundos.

Tabla 5.24: Configuraciones del modelo con el algoritmo Regresión Logística

C	max_iter	media	Desviación típica	tiempo
1	10	0.629	0.026	0.204
1	50	0.692	0.019	0.243
1	100	0.693	0.019	0.344
1	200	0.693	0.018	0.370
1	300	0.693	0.018	0.459

Una vez obtenida la matriz de confusión (Tabla 5.25) con la mejor configuración del modelo, se calculan las métricas (Tabla 5.26) para poder examinar así, de una forma más exhausta, el resultado alcanzado.

Tabla 5.25: Matriz de confusión del test con el algoritmo Regresión Logística

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	582	205	86
Neutro	125	1015	96
Positivo	98	272	518

Tabla 5.26: Resultados de las métricas para el test con el algoritmo Regresión Logística

Métrica	Negativo	Neutro	Positivo	Media
Precisión	72,30 %	68,03 %	74 %	71,44 %
Recall	66,66 %	82,12 %	58,33 %	69,04 %
Accuracy	70,57 %			

Al comparar los resultados con los obtenidos en (Holgado et al. 2020) se observan un *accuracy* y precisión muy parecidos. Sin embargo el *recall* ha empeorado, seguramente debido al aumento de clases. Con todo esto se puede considerar que el modelo obtenido generaliza adecuadamente sin llegar a sobreajustar.

5.4.8. Evaluación del Algoritmo Clasificador de Vector de Soporte

El parámetro que ha resultado ser el más importante del SVC ha sido el C, el cual representa la regularización del modelo. El tiempo promedio de ejecución ha sido 0.0202 segundos. A partir de ahí, los resultados finales de la Tabla 5.27 no han variado demasiado en cuanto a rendimiento (como se puede comprobar con la media), y solamente se ha alterado un poco la consistencia cuando se han variado el número de iteraciones máximas, como se puede comprobar en la desviación típica.

Tabla 5.27: Configuraciones del modelo con el algoritmo SVC

C	max_iter	Media	Desviación típica	Tiempo
1	5	0.722	0.018	0.060
1	30	0.722	0.018	0.142
1	40	0.722	0.018	0.125
1	50	0.722	0.020	0.164
1	1000	0.722	0.018	0.519

Se ha realizado la matriz de confusión (Tabla 5.28) con la mejor configuración y posteriormente se han calculado las métricas (Tabla 5.29) para comprobar la calidad de los resultados.

Tabla 5.28: Matriz de confusión del test con el algoritmo SVC

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	625	176	72
Neutro	123	1047	66
Positivo	103	251	534

Al comparar los resultados con los de (Holgado et al. 2020), se puede observar que han mejorado, el *accuracy* ha aumentado pasando de 70 % a 73,6 %. Además, de media se obtienen valores de *precision* y de *recall* superiores. Se pasa de tener un 67,9 % en precision y un 74,7 % en *recall* a tener de media un 78 % y un 72,1 % respectivamente.

Tabla 5.29: Resultados de las métricas del test con el algoritmo SVC

Métrica	Negativo	Neutro	Positivo	Media
Precisión	73,44 %	71,03 %	79,46 %	74,43 %
Recall	71,59 %	84,71 %	60,13 %	72,15 %
Accuracy	73,60 %			

5.4.9. Evaluación del Algoritmo Bayes Ingenuo de Modelos de Bernoulli Multivariados

En los resultados de las mejores configuraciones de este algoritmo en el *train*, se pueden observar en la Tabla 5.30 como el *accuracy* es algo bajo al superar levemente el 60 %, aunque en contrapartida tiene una buena varianza. Además, ha sido uno de los algoritmos con menor tiempo promedio de ejecución, siendo este 0.0118 segundos.

Tabla 5.30: Configuraciones del modelo con el algoritmo Bernoulli Naive Bayes

binarize	fit-prior	Media	Desviación típica	Tiempo
0.13	False	0.616	0.016	0.009
0.14	False	0.618	0.018	0.009
0.15	False	0.617	0.014	0.010
0.19	False	0.607	0.015	0.009
0.2	False	0.602	0.013	0.009

Con los modelos alcanzados de las anteriores configuraciones se ha obtenido las matrices de confusión (Tabla 5.31) ante el subconjunto de *test*, eligiendo únicamente las de mejor resultado; considerando en este caso la configuración 4.

Tabla 5.31: Matriz de confusión del test con el algoritmo Bernoulli Naive Bayes

Real \ Predicción	Negativo	Neutro	Positivo
Negativo	583	181	109
Neutro	114	1002	120
Positivo	99	259	530

Al comparar los resultados con los obtenidos en (Holgado et al. 2020) se obtienen unos resultados muy similares tanto en *accuracy* como en precisión, aunque con unos resultados peores en *recall*. Teniendo en cuenta que tiene una categoría más a la hora de clasificar, se considera un éxito el modelo obtenido. Se obtienen a través de la la matriz de confunsión las métricas alcanzadas por este modelo (Tabla 5.32).

5.4.10. Comparación de Resultados y Análisis de Tiempos

Se muestra en la Tabla 5.33 una comparación de resultados obtenidos después de realizar todos los experimentos. Después de realizar estos experimentos se llegó a la

Tabla 5.32: Resultados de las métricas del test con el algoritmo Bernoulli Naive Bayes

Métrica	Negativo	Neutro	Positivo	Media
Precisión	72,97 %	69,06 %	69,74 %	70,59 %
Recall	66,78 %	81,00 %	58,89 %	68,89 %
Accuracy	70.27 %			

conclusión de que el algoritmo SGD obtiene los mejores resultados. Se realizó una comparativa entre los trabajos de la Sección 4.1.

Tabla 5.33: Resultados de los diferentes algoritmos propuestos

Algoritmo	Accuracy	P. Neg	P. Neu	P. Pos	P. Media	R. Neg	R. Neu	R. Pos	R. Medio
SGD	75,94 %	79,27 %	73,91 %	76,41 %	76,53 %	70,10 %	86,65 %	66,78 %	74,51 %
XGBoost	74,41 %	77,67 %	72,37 %	74,70 %	74,92 %	72,51 %	83,09 %	64,19 %	73,26 %
KNN	62,46 %	56,71 %	66,62 %	63,80 %	62,37 %	71,13 %	71,84 %	40,88 %	61,28 %
Árbol de decisión	49,52 %	46,05 %	56,32 %	40,00 %	47,46 %	42,73 %	65,94 %	33,33 %	47,33 %
Bosques Aleatorios	63,59 %	67,24 %	60,12 %	70,76 %	66,04 %	58,30 %	86,00 %	37,61 %	60,64 %
Multinomial NB	74,34 %	71,93 %	78,85 %	70,96 %	73,91 %	77,78 %	74,51 %	70,72 %	74,34 %
Regresión Logística	70,57 %	72,30 %	68,03 %	74,00 %	71,44 %	66,66 %	82,12 %	58,33 %	69,04 %
SVC	73,60 %	73,44 %	71,03 %	79,46 %	74,43 %	71,59 %	84,71 %	60,13 %	72,15 %
Bernoulli NB	70.27 %	72,97 %	69,06 %	69,74 %	70,59 %	66,78 %	81,00 %	58,89 %	68,89 %
Media	68,30 %	68,62 %	68,48 %	68,87 %	68,63 %	66,40 %	79,54 %	54,54 %	66,83 %

Como se puede comprobar en la Figura 5.4, el algoritmo Bosques Aleatorios tarda mucho más que los otros. Esto se debe al propio funcionamiento del algoritmo ya que necesita entrenar una serie de árboles para luego hacer una selección por votación. El número de árboles se decide en el parámetro *n_estimators*.

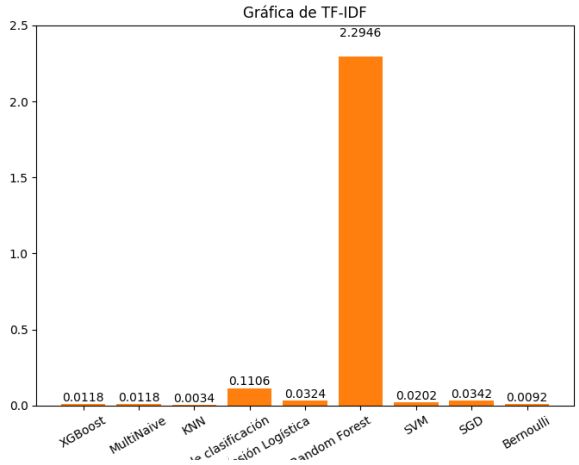


Figura 5.4: Gráfica de la media de los tiempos por clasificador usando conjunto de *train*

En la Tabla 5.34 se pueden ver los *accuracy* promedio de los trabajos estudiados previamente. Se podría decir que comparado con los resultados de otros proyectos; el

que proponemos se queda muy por detrás. También va a influir el número de algoritmos usados, ya que es mucho mayor así como el tipo de preprocesado y el conjunto de datos utilizado.

Tabla 5.34: Tabla comparativa

Referencia	Algoritmos	Etiquetas	Idioma	Resultados
(SAURA et al. 2018)	Monkeylearn	3	Inglés	
(Martin-Domingo et al. 2019)	Theysay y Twinword	32	Inglés	P = 78.7 % ACC = 69.6 %
(Sabariah et al. 2015)	Diccionario	2	Inglés	80 %
(Cernian et al. 2015)	SentiWordNet	3	Inglés	éxito=61 %
(Nguyen et al. 2018)	LLR, SVM, aumento de gradiente, VADER, Pattern,SentiWordNet	3	Inglés	LLR=90 % VADER=83 %
(Joyce & Deng 2017)	NB	2	Inglés	éxito=94 %
(Go et al. 2009)	NB, Maximum Entropy y SVM	2	Inglés	80 %
(Agarwal et al. 2011)	unigram, basado en características, kernel de árbol y SVM	3	Inglés	75.39 %
(Wan & Gao 2015)	SVM, NB, Arboles de Decisión C4.5(DTree), RF y un clasificador ensamblado	3	Inglés	83.5 %
(Buntoro et al. 2016)	NB	5	Inglés	P=71,9 %
(Loyola-González et al. 2019)	Bayesian Network, KNN, C4.5, NB, RF, SVM, LLR, Adaptive Boosting, Bagging, PBC4ci, Multilayer Perceptron		Inglés	ACC= 90 %
(Sproġis & Rikters 2020)	Naive Bayes y Perceptrón	3	Inglés	61,23 %
(Janssens et al. 2015)	SGD*modified hubber*,SGD *hinge*,SGD *log*, SGD *perceptron*,LIBLINEAR, MNB, BNB,*Ridge*, Nearest centroid	2	Inglés	ACC=66,64 %
(Holgado et al. 2020)	NB original, Naive Bayes multinomial, Naive Bayes Bernouilli, Regresión logística, SGD y SVC	2	Castellano	72,32 %
(Hurtado et al. 2015)	SVM	6	Castellano	ACC = 67.3 %
(RODRíGUEZ 2018)	SVM, arbol de decisión y Naive Bayes	2	Castellano	80 %
(Becerra 2017)	Máquina de soporte vectorial, árbol de decisión y Naive Bayes multinomial	2	Castellano	ACC = 87.15 %
Modelo propuesto	SGD, XGBoost, KNN, Árboles de clasificación, Bosques Aleatorios, Clasificador Bayesiano Ingenuo Multinomial, Clasificador Bayesiano Ingenuo de Bernoulli, Máquina de Soporte Vectorial y Regresión Logística	3	Castellano	ACC=68,30 %

Por otro lado se puede observar como en el trabajo que se ha tomado como referencia a mejorar y del que se ha tomado el dataset, solo baja 4 puntos el *accuracy* promedio, aún habiendo aumentado el número de clases a clasificar.

Tabla 5.35: Tabla comparativa trabajos más relacionados

Algoritmo	(Holgado et al. 2020)	(RODRíGUEZ 2018)	(Becerra 2017)	Modelo propuesto
SGD	71.14 %	X	X	75.94 %
XGBoost	X	X	X	74.41 %
KNN	X	X	X	62.46 %
Árboles de clasificación	X	80 %	66.85 %	49.52 %
Bosques Aleatorios	X	X	X	63.59 %
Clasificador Bayesiano Ingenuo Multinomial	72.24 %	X	84.60 %	74.34 %
Clasificador Bayesiano Ingenuo de Bernoulli	72.80 %	X	X	70.27 %
Máquina de Soporte Vectorial	70.45 %	X	87.15 %	73.60 %
Regresión Logística	71.88 %	X	X	70.57 %

Capítulo 6

Contribuciones Individuales

Este Trabajo de Fin de Grado surge como idea de nuestros tutores académicos. Los cuatro alumnos estábamos interesados en aprender e investigar sobre la Inteligencia Artificial y el Análisis de Sentimientos. El poder haber hecho un trabajo entre cuatro personas nos ha permitido experimentar con un elevado número de clasificadores y trabajar sobre sus métricas. Se ha podido repartir el trabajo equitativamente, intentando aportar cada uno lo mejor de lo que le habían enseñado durante nuestro desarrollo universitario. Nuestra intención fue aportar una nueva clase neutra en un dataset en castellano y comparar la clasificación de análisis de sentimientos a través de varios algoritmos llegando a escoger el que mejores resultados ha conseguido.

Los trabajos que se realizan por varios miembros requieren una constante comunicación entre aquellos que lo forman. Se realizaban reuniones semanales con nuestros tutores y la comunicación entre los cuatro era bastante fluida ya que realizábamos reuniones entre nosotros al menos una vez entre semana, lo que ha permitido ayudarnos entre nosotros y en muchas ocasiones aprovechar mejor el tiempo. A continuación se va a explicar con más detalle lo que ha aportado cada miembro del equipo para realizar dicho proyecto y todos los conocimientos que se han ido adquiriendo.

6.1. Rodrigo Fernández Ambrona

El año previo al TFG cursé la asignatura de Inteligencia Artificial del itinerario de Computación de Ingeniería Informática. Tras elegir el TFG, realicé dos cursos de Coursera por recomendación de los tutores Launching into Machine Learning y Natural Language Processing in TensorFlow, los cuales me dieron un conocimiento más profundo sobre el tema. Estos conocimientos previos me ayudaron a ver el conjunto del problema al que nos enfrentábamos, así como saber en qué temas había que enfocarse y dedicarle mas tiempo. También como preparación previa tuve que adquirir unos conocimientos básicos de LaTeX en el manejo de referencias, enumeraciones, creación de tablas y demás.

Una vez empezado el TFG, busqué diferentes *papers* y otros documentos de interés sobre los PLN y sus diferentes algoritmos a través de *Google Scholar*, intentando buscar trabajos recientes y con corpus en español aunque acabé mirando también corpus en inglés. Estos trabajos nos servirían más adelante para la realización del estado del arte así como en la toma de decisión de qué algoritmos elegir para nuestro proyecto. Al final de esta primera etapa de investigación empecé a preparar el preprocesado del corpus. Elegimos un preprocesado basado en palabras y no en sintaxis, por lo que tuve investigar las expresiones regulares para la eliminación de palabras que cumplían ciertos patrones como es el caso de los *links* o las menciones que empiezan siempre igual. También investigué distintos diccionarios de palabras de parada en español, eligiendo finalmente el diccionario dado por la librería de NLTK. Al final, el preprocesado consistió en la lectura de documentos svc, limpieza y corrección de datos así como eliminación de palabras de parada y almacenamiento de la aplicación iterativa de los diferentes filtros en nuevas columnas del propio svc donde se almacenaba el corpus. En esta parte me encargué principalmente de la limpieza, es decir, la eliminación de menciones, links, palabras de parada, normalización del texto a UTF-8 con caracteres en minúscula y eliminación de repeticiones de tweets. La parte de corrección ortográfica fue realizada por una de mis compañeras, aunque yo la localicé dentro del preprocesado. Una vez hecho el preprocesado realicé algún estudio y visualización de datos longitud final de los tweets y palabras más repetidas así como una visualización aleatoria para detectar pequeños fallos como perdida de significado por eliminación de ciertas palabras, modificando acordemente las palabras de parada. Posteriormente redactaría de forma conjunta la parte correspondiente de la memoria. También participé en el Capítulo 2 con la explicación del funcionamiento de los diferentes algoritmos que íbamos a usar en los experimentos, así como en enumera y explicar los diferentes parámetros que podían usarse. También redacté el último punto sobre el sobreaprendizaje.

Tras finalizar esta parte, se decidió incorporar una tercera categoría a predecir. Se usaron varios etiquetadores y ayudé a la codificación, ejecución, pues y solución de problemas. Aunque no se pudo reducir tiempos sí logré su completa automatización. Después del etiquetado diseñé el sistema para medir cuál etiquetado nos aportaba mejores resultados y, en consecuencia, a elegir el mejor. Tras los experimentos de etiquetado realicé la parte correspondiente en la memoria. En la parte de experimentos de los diferentes algoritmos con el conjunto de entrenamiento, realicé la optimización de parámetros de los dos algoritmos que me fueron asignados así como la tabulación de sus resultados. Una vez obtenidos todos los resultados de test de los diferentes algoritmos participé activamente en el proceso de decisión de la mejor configuración con cada algoritmo, la tabulación y justificación de sus resultados y posterior comparación con los resultados obtenidos en otras investigaciones. Esas han sido mis aportaciones principales, aunque cabe destacar que durante todo el proceso hemos estado revisando y ayudándonos los unos a los otros en la redacción de las memoria así como con la parte de los experimentos.

6.2. Cristina Molina Gerbolés

En el curso 2019/2020 cursé la asignatura de Inteligencia Artificial. Antes de empezar este trabajo realicé los cursos siguientes cursos: *Launching into Machine Learning, How Google does Machine Learning* y *Natural Language Processing in TensorFlow* que ofrecía coursera, me sirvieron para poder comprender como funciona la librería TensorFlow para el PLN además de darme una visión mucho más amplia sobre la inteligencia artificial y el aprendizaje automático, ampliando y complementando mis conocimientos en este área. Posteriormente, cuando la línea de trabajo estaba más definida, me formé más sobre la materia y el tema a experimentar, en este caso el análisis de sentimientos usando PLN. Realicé una revisión del estado del arte usando *Google Scholar*, ordenamos esos trabajos para estudiarlos. Posteriormente, definimos como debería implementarse nuestro modelo, las fases que tendría, los algoritmos que se iban a utilizar y la forma de preprocesado que se seguiría. Todo esto se concretó en las reuniones semanales que fuimos teniendo a lo largo del año con nuestros tutores. Para tener una interfaz parecida y no tener problemas a la hora de códificar se utilizó *Pycharm* como IDE de desarrollo y complicación de código, por otra parte se utilizó el repositorio en *Gitlab* del grupo GASS para tener todo el código en la nube y poder trabajar en remoto, además de llevar un control de versiones por si algo fallaba, detectarlo de forma eficiente.

El primer paso a realizar fue hacer el preprocesado y el etiquetado, en esta parte era crucial corregir el texto ya que contamos con un conjunto de datos sacado de una red social, lo que podría llevarnos a tener bastantes errores gramaticales y por lo tanto que la máquina no detectara bien el texto, generando así un mal diseño. Para esta tarea necesité decidir que metodología iba a usar, tenía dos opciones, usar un algoritmo que tradujera teniendo en cuenta un texto de referencia, por ejemplo un libro o usar librerías que pudiera proporcionarme *python*. Ambas opciones tienen sus ventajas y desventajas, aunque a la hora de elegir pesó más la eficiencia en tiempo y decidí usar una librería de *python*.

El siguiente paso fue encontrar la librería a usar, me decidí por *Hunspell* ya que era la que mejores correcciones ofrecía en castellano. Después de todo el preprocesado toca hacer el etiquetado del texto, en una de las reuniones semanales se nos aconsejó usar *Textblob* y *Vader* para poder hacerlo, el problema fue que no admitían el texto en castellano así que hubo que buscar la manera de traducir el texto y decidir si lo que se perdía de *accuracy* merecía la pena. Una vez montada la estructura del código para el etiquetado hubo que etiquetarlo y al tener problemas con el servidor decidí probar a hacerlo a mano, pasando de 300 en 300 todo el corpus por el etiquetador. En paralelo a este proceso se fue escribiendo la memoria en *LaTeX* para que no se olvidara ninguna de las complicaciones y características obtenidas. Más adelante tuvimos que replicar un código de un trabajo anteriormente realizado, para poder entender mejor los clasificadores utilizados. Además tuvimos que decidir que clasificadores íbamos a usar para nuestro modelo. Una vez decididos los clasificadores y montado el `GridSearchCV`, nos repartimos los clasificadores

a ejecutar para poder realizar la extracción de datos de una forma más eficiente. Tuve que indagar sobre los árboles de clasificación y sus hiperparámetros, sobre el XGBoost y sobre la regresión lineal aunque luego no se ha llegado a usar para las pruebas finales. Tras esto, escribí el capítulo 2 teniendo en cuenta todos los conocimientos obtenido de las fases anteriores, además de tabular todos los resultados obtenidos en la fase de entrenamiento. Además realicé la mayoría de imágenes que aparecen en la memoria.

A la par que esto sucedía, se revisaron nuevamente los trabajos estudiados del estado del arte para documentarlos en el capitulo 4. A la hora de realizar la comparación de tiempos y resultados obtenidos con diferentes muestras y diferentes algoritmos de clasificación, realicé un programa en *python* para ayudarme a graficar dichos resultados.

Lo último que faltaba para terminar los experimentos era hacer las pruebas con el conjuntos de *test*, por lo que ayudé a mi compañero a tabular los resultados que iba obteniendo de la ejecución. Además de elegir las mejores configuraciones que iban a plasmarse finalmente en la memoria.

Con los resultados tabulados se realizó su análisis, esto fue un trabajo a veces difícil puesto que hay algunos algoritmos de los que elegimos que son de caja negra por lo que no se puede saber muy bien como funcionan realmente por dentro. Tras terminar todo esto se hizo un repaso a la memoria ya que había cosas que escribimos muy al principio y que no se ajustaban a lo que estábamos haciendo en ese momento por lo que tuvimos que reescribir algunas partes además de reorganizar la memoria en diferentes capítulos. Debido a que durante desarrollo del proyecto teníamos diferentes disponibilidades de tiempo hubo que repartir y gestionar el tiempo y el trabajo que hacía cada uno, además los roles adquiridos por los miembros del equipo fueron rotando bastante, algo que me ayudó a mejorar mi faceta como líder y de trabajo en equipo.

6.3. Patricia Motoso González

Inicialmente, cuando se comenzó a idear este proyecto en el mes de Septiembre, se me ofreció estudiar, por parte de los profesores, los siguientes cursos que ofrecía Coursera: *Launching into Machine Learning*, How Google does Machine Learning y *Natural Language Processing in TensorFlow*. Debido a que me incorporé más tarde al proyecto el acceso a estos cursos ya no eran gratuitos, por lo que me puse a investigar por mi cuenta, gracias al buscador de Google o tutoriales en You Tube, todo lo relacionado con la librería *Python* y como se manejaba ya que nunca había tenido que hacer uso de ella en otras asignaturas. Por otro lado investigué como funcionaba la librería TensorFlow para el PLN y me leí artículos con el que pude entender mejor en que consistía la Inteligencia Artificial y el aprendizaje automático, campo que era completamente desconocido para mí.

Un poco más adelante, cuando la línea de trabajo estuvo más definida, me formé un poco más en el tema que se iba a experimentar, en este caso el análisis de sentimientos usando PLN. Para ello realizamos entre todos los compañeros una búsqueda de trabajos

en *Google Scholar*, y lo clasificamos en una tabla para poder utilizarlos más adelante.

Utilizamos la interfaz *Pycharm* como IDE de desarrollo, y el repositorio *Gitlab* del Grupo GASS donde se tenia el código en la nube y se podía trabajar en remoto entre todos los compañeros desde su propia casa.

Mientras parte del grupo se dedicaba al preprocesado del dataset, tarea que me resultó bastante complicada, ya que trabajaba sobre máquina virtual lo que ralentizaba el ordenador por lo que no era capaz de generar los datos correctamente; yo me dedique a realizar parte del capítulo 2 relacionado con el lenguaje Natural y el Análisis de Sentimientos, que luego fue transformado en el capítulo 3.

Empecé a iniciarme con el editor de *LaTeX*, ya que aunque teníamos una plantilla inicial para ir trabajando sobre ella, nunca había utilizado dicho editor. Empecé a mirar en tutoriales de internet como se realizaban tablas e imágenes que luego incorporaríamos en nuestra memoria.

Por otro lado, a la par que escribía en dicho editor el capítulo 2, me dedique a buscar trabajos que estuvieran relacionados con los que estábamos investigando, para poder después documentarlo en la memoria en el capítulo 4, concretamente el estado del arte.

Después de realizar esta investigación, era necesario terminar los experimentos realizando las pruebas con el conjunto de *test*. Fuí ayudando a mis compañeros a tabular los resultados que se iban obteniendo de la ejecución. A parte se eligió las mejores configuraciones que más tarde se plasmarían en la memoria.

Una vez tuvimos los resultados ya tabulados nos dispusimos a comentarlos, dividiéndonos el trabajo entre los 4 miembros. Sin embargo, debido a que empezaron a surgir errores en la memoria, a la vez que realizábamos las tabulaciones, me dispuse a arreglar dichos errores que nos iban indicando los profesores, desde tablas a párrafos mal redactados o bibliografía.

Se habían escrito cosas muy al principio y al final no se ajustaban a lo que estábamos haciendo en ese momento por lo que se tuve que reescribir algunas partes además de reorganizar la memoria en diferentes capítulos.

Ha sido un proyecto el que no teníamos las mismas disponibilidades de tiempo. Por ello se tuvo que repartir y gestionar el tiempo y el trabajo que hacía cada uno, además los roles adquiridos por los miembros del equipo fueron rotando bastante, algo que me ayudó como gestionar a veces un grupo de trabajo. Cada uno ha podido aportar parte de lo que se le da mejor tanto a nivel de código como de documentación y ha sido tratado de forma consensuada a través de reuniones semanales.

6.4. Juan Antonio Carrión García

Previo al TFG cursé la asignatura Inteligencia Artificial del itinerario de Ciencias de la Computación de Ingeniería Informática, la cual seguramente fue la más importante a la hora de entender qué es lo que se iba haciendo durante el transcurso del mismo, y durante

dicho transcurso, he estado cursando las asignaturas Cloud y Big Data, la cual me dio más amplitud a la hora de ver formas de procesar datos en paralelo, y Minería de Datos, la cual aportó perspectiva a la hora de medir rendimientos de clasificadores.

Dado que me incorporé más tarde al proyecto, el acceso a los cursos facilitados se complicó dado que en el momento en el que llegué al proyecto ya no eran gratuitos. Esto me impulsó a buscar por mi cuenta diversos trabajos de GitHub que tratasen de la temática Aprendizaje Automático, aparte de investigar acerca de los papers que se nos iban facilitando. El hecho de ver tantos proyectos y de leer mucho código me aportó mucho criterio, y sirvió de ayuda más adelante a la hora de programar toda la parte relacionada con los experimentos y de hacer un código limpio, mantenible, escalable, todo ello echando mano de tecnologías como la Programación Orientada a Objetos, o diversos patrones aplicados al Aprendizaje Automático, como bien puede ser el caso del Pipeline.

Durante la primera parte del proyecto, en la que se realizó el preprocesado, trabajé en la redacción del Capítulo 2, tanto redactando gran parte del mismo. Más adelante, compartí apuntes propios de clasificadores a los compañeros para ayudarles en la tarea de completar la explicación teórica de dichos clasificadores.

Pero mi principal aportación al proyecto, sin duda, es el código con el que se realizaron los experimentos. Tan pronto como el *dataset* terminó de ser preprocesado, se comenzó el desarrollo de un código que facilitase la experimentación automática de múltiples combinaciones de hiperparámetros, a la vez que tuviera la capacidad de poder añadir nuevos clasificadores sin necesidad de entrar en la implementación que había por debajo. La elaboración de este código fue costosa y llevó muchas semanas. Otro problema que se enfrentó fue conseguir que los experimentos fueran reproducibles, es decir, que los clasificadores diesen siempre el mismo resultado, y demostrar que dicha combinación de hiperparámetros sería siempre la mejor. Gracias a diversas mejoras del código y a la prueba de varias combinaciones de hiperparámetros, se consiguió un rendimiento superior en la mayoría de los clasificadores, como se puede comprobar en los resultados del Capítulo 5. Por último, cabe mencionar que la elaboración del código me exigió el aprendizaje de tecnologías con las que no estaba familiarizado, como bien puede ser Python Orientado a Objetos, o una comprensión profunda del funcionamiento interno de las librerías usadas.

El método de trabajo a la hora de ir construyendo el código era el siguiente: Cada semana se iban pidiendo ẗitosÿ se comprobaba que el código hacía exactamente lo que requería. En caso de estar todo correcto, se procedía con la siguiente tarea, y en el caso de haber fallos, se procedía a cambiar lo que estuviera erróneo o impreciso a la hora de ejecutar, y se procedía con la tarea de la siguiente semana. Esto conllevó sustanciales cambios a la hora de que se fuera haciendo el código. Como por ejemplo: Trabajar sobre una matriz TF-IDF con un vocabulario extraído por la propia matriz, o con un vocabulario optimizado con las 2000 palabras más importantes del corpus.

Una vez se terminó el código, tabulé, junto a mi compañera, los resultados de los experimentos y formé parte del análisis de dichos experimentos. Por último, hice la

descripción del modelo en el Capítulo 5, la Introducción, y el Resumen.

Además de las partes de la memoria que redactamos, tanto mis compañeros como yo, revisamos aquellos fragmentos que no habíamos escrito, lo cual facilitó mucho la búsqueda de pequeños errores de redacción, o de imprecisiones a la hora de definir algún término.

Capítulo 7

Conclusiones y Trabajo Futuro

En este capítulo se van a detallar en la Sección 7.1 los conclusiones a las que se han llegado después de realizar los experimentos y en la Sección 7.2 los cambios que se podrían realizar en el futuro y que mejorarían varios aspectos del trabajo.

7.1. Conclusiones

En este trabajo de investigación se partía de la idea de desarrollar un analizador del sentimiento de llamadas de audio en español. Ante la inexistencia de este tipo de *corpus*, se obtó por usar un *corpus* de mensajes de texto con contenido científico extraídos de Twitter con la intención de encontrar un modelo de clasificar el sentimiento en tres categorías (positivo, neutro y negativo). Dejando para futuras investigaciones la extrapolación de los resultados a un corpus más similar al de llamadas de audio.

Para los analizadores de sentimientos se usaron diferentes algoritmos de clasificación (Descenso del Gradiente Estocástico, Aumento de Gradiente Extremo, K Vecinos más Cercanos, Árboles de clasificación, Bosques Aleatorios, Clasificador Bayesiano Ingenuo Multinomial, Clasificador Bayesiano Ingenuo de Bernoulli, Máquina de Soporte Vectorial y Regresión Logística) y se usa la precisión, el *accuracy* y el *recall* como métricas para medir el desempeño de los diferentes modelos y poder compararlos.

Con los resultados obtenidos en la Sección 5.4 se puede observar que la clase que peor se distingue es la positiva, esto se debe al etiquetado que realizamos a través de VADER. Éste necesitaba de un texto en inglés para poder etiquetar, por lo que se decidió usar una API para traducir los *tweets*. Con este proceso se asumió una pérdida de fiabilidad al etiquetar. Observando el *corpus* etiquetado, se aprecia que tendió a etiquetar como positivos algunos *tweets* que son neutros o claramente negativos, esto complica la tarea de clasificar. En yuxtaposición, la clase neutra fue la que mejor se etiquetó potencialmente debido al carácter científico de los *tweets*, que tienden a ser neutros. En consecuencia, el bajo *recall* de la clase positiva se puede deber en gran medida a este fenómeno.

Indistintamente del clasificador se observa un alto *recall* en la clase neutra, la cual

tiene más representación que las demás en el *corpus*. Esto podría indicar un sesgo de los datos reflejado en la mayoría de los algoritmos a excepción del clasificador bayesiano ingenuo multinomial. Sin embargo se observa como la precisión mantiene unos valores estables en todos los modelos, manteniendo un valor similar a su vez al *accuracy*. Esto puede indicar que el modelo que se usa es consistente y que a su vez no sobreaprende, ya que los resultados tanto en *train* como en *test* son similares, además de no tener una alta varianza.

Por otro lado se advierte una clara deficiencia en cuanto al campo de investigación en castellano u otros idiomas. La mayoría de trabajos realizados se encuentran en inglés y la mayoría de etiquetadores eficientes solo entienden el inglés.

Se ha introducido una nueva categoría a la hora de clasificar el conjunto de datos, esto se ha realizado a través de un etiquetado automático. Para lograrlo se ha debido de recurrir a una previa traducción automática, puesto que los etiquetadores convencionales necesitan de texto en inglés. Todo ello ha supuesto un pequeño descenso del *accuracy* respecto al trabajo base, en este se obtiene 72,32 % mientras que en el modelo propuesto se obtiene 68,30 % de promedio, una disminución de 4 puntos. Se puede considerar una mejora debido a que se ha aumentado el nivel de información extraída de los datos. Otro de los objetivos logrados ha sido el uso de un preprocesado propio basado en palabras en lugar de un significado semántico como se venía usando en el trabajo original. Pese al cambio no se observan grandes diferencias en los resultados de preprocesado aunque se considera más eficiente.

7.2. Trabajos futuros

Como posibles trabajos futuros se podrían mejorar los siguientes aspectos:

- Añadir al preprocesado del *corpus*, la eliminación de los sufijos para normalizar el texto.
- Usar el algoritmo *word2vec* para transformar los datos después de realizar el etiquetado. Utiliza un modelo de red neuronal para aprender asociaciones de palabras de un *corpus* con gran cantidad de texto. Dicho algoritmo, una vez que se entrena, puede detectar palabras sinónimas.
- Usar un etiquetador que no sea necesario traducir al castellano para conseguir mejorar la clase positiva.
- Usar un *corpus* basado en llamadas en castellano.
- Tener un servidor, que al traducir el corpus para el etiquetado, este preparado para afrontar un número elevado de peticiones sin que llegue a bloquearse.

Resumen del Trabajo en Inglés

Capítulo 8

Introduction

8.1. Introduction and context

Nowadays, there are more than 353 millions active users on Twitter. It is a social network based on public opinion about topics of relevance, so it becomes extremely useful when you are trying to getting an overview of public opinion.

During the last years, there has been an exponencial growth on the number of people using these type of platforms, as it is shown in the Figure 8.1, whose function is to send opinions about any topic in messages of limited length. Impressed by that growth, big companies got interested and started researching ways to mine and extract data from Twitter in order to obtain the most detail information about people's opinions.

So, in this context there are two Artificial Intelligence (AI) branches that can be exploited. One of them is the Sentiment Analysis (SA) and the other is Natural Language Processing (NLP). Some of the applications that can be given could be doing a market analysis or simply getting to know the opinion of the customers about a product or service they hired. All these applications would give fantastic results and predictions that are very close to reality, due to the fact that the processing capability of the computer has increased, and large amounts of data can be processed in a reasonable time.

On the other hand, one of the advantages of these two Machine Learning research fields, is the fact that they can be used in a large amount of problems related to word processing. A classifier trained with Sentiment Analysis and Natural Language Processing techniques to predict Tweets, could be used to solve other type of problems, such as a Call Analysis, as long as that classifier had been trained with a Corpus with enough samples.

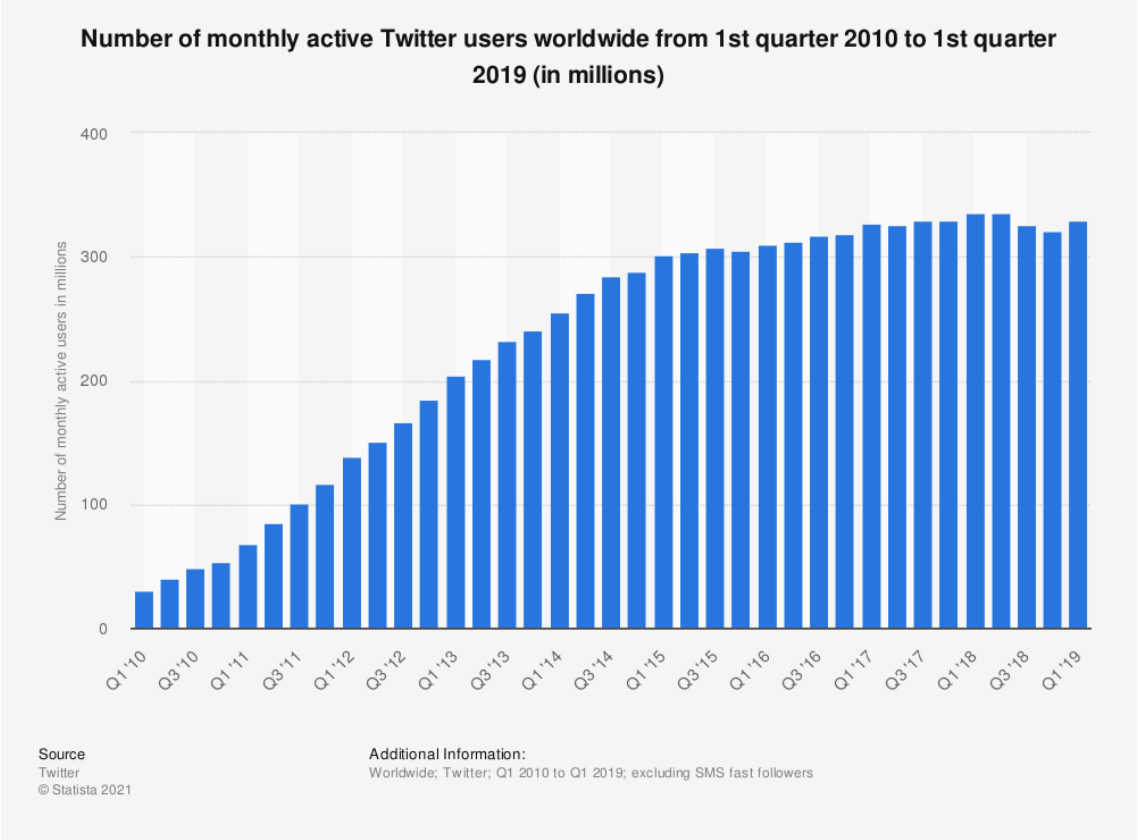


Figura 8.1: Graphic of the evolution of Twitter users

8.2. Context

This Final Degree Project has been carried out within the Analysis, Security and Systems Group (GASS Group, https://gass.ucm.es/, Group 910,623 from the catalog of groups recognized by the Complutense University of Madrid) as part of the research project activities THEIA (Techniques for Integrity and Authentication of Multimedia Files of Mobile Devices) with reference FEI-EU-19-04.

8.3. Goals

On this proyect, the work (Holgado et al. 2020) is taken as reference. The main objective is to improve their research results, adding some new categories to extract more detailed information about the data, and also getting the best algorithm in order to extract tweets sentiment. The next main objetive to take into account was to change the preproccesing mechanism in order to improve results, at the same time as using two different tools (VADER and TextBlob) to label automatically the corpus, checking which one performs better.

Considering the set of n most relevant words as the vocabulary in order to classify a

tweet. Another objective of this project is to obtain the optimous number of words that metioned vocabulary must have in order to train the clasiffiers.

In addition an increase in the number of clasification algorithms used in the referential work is proposed, rising the number of clasifications to compare, as well as adding some techniques that are at the vanguard of the machine learning, as neural networks.

8.4. Workplan

This section explains the work phases that have taken place thoughout the entire project.

1. **Investigation**: The first phase of the State of Art was carried out during a period of approximately 4 months, in which papers, documents, and work related to the project were researched. All these had in common the subjects, Natural Language Processing and Tweet Analysis. Despite there were some researched papers whose topics were more related to audio processing and transformation from audio to text, than to Machine Learning, finally that part of the project was delegated to other group, who was in charge of the TFG "Call Analysis II". From then on, the research focused on Natural Language Processing. It is important to note that in this phase there were found all the documents that will be explained later in the chapter 5. Finally, in this phase the programming language Python was studied, and also all libraries and environments that were necessary when implementing the code to make the experiment *Scikit-learn, PyCharm,...*, were downloaded.

2. **Development**: In the second phase, the implementation of the code started. A dataset of tweets was preprocessed until getting a set of data that the classifiers would be capable of using when being trained. Inicially, the tweets were only tagged into two categories: Positive and negative. A new category was added, the neutral category. Some of the tweets from the set were tagged into neutral tweets. Moreover, a dataset processing was made so that from then on, the classifiers didn't have to deal with inherent problems from the data, such as deleting special characters, hashtags and mentions, and words that gave no extra meaning (stop words), and normalizing the text. Once the data were processed, a model was designed to ease the classifier testing.

3. **Experiments**: During the experiment phase, the first thing that was done was checking which was the optimal number of words to separate the text in categories. After getting that number, the entire data of the corpus was transformed to numeric frequencies. And finally, with that frequencies, all the representative combinations of the hyperparameters of the classifiers were tested, with the goal of having a large amount of results and being able to choose wisely which ones were the best. Those classifiers were trained with the train partition of the data.

4. **Results**: Finally, the results to check which classifiers were the best were obtained. It was checked that those classifiers were giving a similar performance with the test partition, therefore, they had not been overfitted or underfitted. From there, conclusions were extracted, comparing our results with the ones from similar investigations.

8.5. Work Structure

This section explains how is the project organized. It consists on 5 chapters.

- On chapter 2 are explained some introductory concepts about data science as well as basic concepts about machine learning. On this chapter is also treated the diferente clasificator that are going to be used during the project. Finally the metrics used to measure out the quality of the clasifications are analized in addition to the overfit problem and how to prevent it.
- On chapter 3 is entirely based on the Natural language procress and sentiment analysis. The differente concepts of PLN that will appear during the project are defined besides some applications of PLN and AS.
- On chapter 4 is anlized the actual condition of the PLN and AS investigation, comparing the various techniques used in different projects.
- On chapter 5 is explained the construction of the used model, as well as all the previous steps that were necessary to reach the classicable corpus. Further the parameters used by the classification algorithms are explainded. Ultimately experiments made along with conclusions are examined.
- At last, on chapter 7 are described the conclusions of all the invetigation proyect as well as the future works.

Capítulo 9

Conclusion

In these chapter it will be described on Section 9.1 the conclusions which has been reached after performin the experiments and Section 9.2 changes that could be made in the future that would improve various aspects of the job.

9.1. Conclusions

In this research work was based on the idea of developing a sentiment analyzer for audio calls in Spanish. In the absence of this type of corpus, it was opted by using a corpus of text messages with scientific content extracted from Twitter with the intention of finding a model to classify sentiment into three categories (positive, neutral and negative). Leaving for future research the extrapolation of the results to a corpus more similar to audio calls.

Different classification algorithms were used for sentiment analyzers (Stochastic Gradient Decrease, Extreme Gradient Increase, K Nearest Neighbors, Classification Trees, Random Forests, Multinomial Naive Bayesian Classifier, Bernoulli Naive Bayesian Classifier, Vector Support Machine and Logistic Regression) and precision, accuracy and recall are used as metrics to measure the performance of the different models and to be able to compare them.

With the results obtained in Section 5.4 it can be seen that the class that differentiate the worst is the positive one, this is due to the labeling we carry out through VADER. This one needed a text in English to be able to tag, so it was decided to use an API to translate the tweets. With this process a loss of reliability was assumed when labeling. Observing the tagged corpus, it can be seen that it tended to tag as positive some tweets that are neutral or clearly negative, this complicates the task of classifying. In juxtaposition, the neutral class was potentially best tagged due to the scientific nature of tweets, which tend to be neutral. Consequently, the low recall of the positive class may be largely due to this phenomenon.

Regardless of the classifier, a high recall is observed in the neutral class, which has more representation than the others in the corpus. This could indicate a bias of the

data reflected in most of the algorithms with the exception of the multinomial Bayesian classifier. However, it is observed how the precision maintains stable values in all the models, maintaining a value similar to the accuracy. This may indicate that the model used is consistent and that it does not overfit, since the results in both train and test are similar, in addition to not having a high variance.

On the other hand, there is a clear deficiency in the field of research in Spanish or other languages. Most of the work done is in English, and most efficient taggers only understand English.

A new category has been introduced when classifying the data set, this has been done through automatic labeling. To achieve this, it has been necessary to resort to a previous automatic translation, since conventional taggers need text in English. All of this has led to a small decrease in accuracy with respect to the base work, in which 72.32 % is obtained, while in the proposed model an average of 68.30 % is obtained, a decrease of 4 points. It can be considered an improvement because the level of information extracted from the data has been increased. Another of the objectives achieved has been the use of its own preprocessing based on words instead of a semantic meaning as it had been used in the original work. Despite the change, no great differences are observed in the preprocessing results, although it is considered more efficient.

9.2. Future work

As possible future work, the following aspects could be improved:

- Add to the preprocessing of the corpus, the elimination of suffixes to normalize the text.
- Use the word2vec algorithm to transform the data after labeling. It uses a neural network model to learn word associations from a corpus with a large amount of text. Such an algorithm, once trained, can detect synonymous words.
- Use a tagger that does not need to be translated into Spanish to improve the positive class.
- Use a corpus based on Spanish calls.
- Have a server that, when translating the corpus for labeling, is prepared to deal with a large number of requests without crashing.

Bibliografía

Agarwal, A., Xie, B., Vovsha, I. & Rambow, Owen Passonneau, R. J. (2011), Análisis de sentimiento de los datos de twitter, *in* 'Actas del taller sobre el lenguaje en las redes sociales (LSM 2011)', pp. 30–38.

Arce, J. I. B. (2019*a*), 'La matriz de confusión y sus métricas', https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/.

Arce, J. I. B. (2019*b*), 'La matriz de confusión y sus métricas', https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/.

Bagnato, J. I. (n.d.), 'Aprendizaje por Refuerzo', https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/.

Becerra, M. (2017), Análisis de sentimientos en twitter: El bueno, el malo y el¿, *in* 'XX Concurso de Trabajos Estudiantiles-JAIIO 46 (Córdoba, 2017).'.

Benoit, K., Watanabe, K., Wang, H., Nulty, P., Obeng, A., Müller, S. & Matsuo, A. (2018), 'quanteda: An r package for the quantitative analysis of textual data', *Journal of Open Source Software* **3**(30), 774.

Betancourt, G. A. (2005), 'Las máquinas de soporte vectorial (svms)', *Scientia et technica* **1**(27).

Biehl, M. & Riegler, P. (1994), 'On-line learning with a perceptron', *EPL (Europhysics Letters)* **28**(7), 525.

Bismart (2021), 'Diferencias entre el Machine Learning supervisado y no supervisado', https://blog.bismart.com/es/diferencias-machine-learning-supervisado-no-supervisado.

Borg, A. & Boldt, M. (2020), 'Using vader sentiment and svm for predicting customer response sentiment', *Expert Systems with Applications* **162**, 113746.

Browne, M. W. (2000), 'Cross-validation methods', *Journal of mathematical psychology* **44**(1), 108–132.

Buntoro, G. A., Adji, T. B. & Purnamasari, A. E. (2016), 'Candidatos de análisis de sentimiento del presidente de indonesia 2014 con atributo de cinco clases', *Revista internacional de aplicaciones informáticas* (136.2), 23–29.

Calvo, D. (2019), 'Aprendizaje supervisado', https://www.diegocalvo.es/aprendizaje-supervisado/.

Cambria, E., Olsher, D. & Rajagopal, D. (2014), Senticnet 3: a common and common-sense knowledge base for cognition-driven sentiment analysis, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 28.

Caparrini, F. S. (2017), 'Introducción al Aprendizaje Automático', http://www.cs.us.es/~fsancho/?e=75.

Carpenter, B. (2007), Lingpipe for 99.99 % recall of gene mentions, *in* 'Proceedings of the Second BioCreative Challenge Evaluation Workshop', Vol. 23, Citeseer, pp. 307–309.

Carreras, X., Chao, I., Padró, L. & sy Padró, M. (2004), Freeling: un conjunto de analizadores de idiomas de código abierto., *in* 'LREC', pp. 239–242.

Cernian, A., Sgarciu, V. & Martin, B. (2015), Sentiment analysis from product reviews using SentiWordNet as lexical resource, *in* '2015 7th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)', IEEE, Bucharest, Romania, pp. WE–15.

Cortez Vasquez, A., Vega huerta, H., Pariona Quispe, J. & Huayna, A. M. (2009), 'Procesamiento de lenguaje natural', *Revista de investigación de Sistemas e Informática* **6**(2), 45 – 54.
URL: *https://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/5923*

Cunningham, H. (2002), 'Gate, a general architecture for text engineering. computers and the humanities', (2).

da Silva, N. F., Hruschka, E. R. & Hruschka, E. R. (2014), 'Tweet sentiment analysis with classifier ensembles', *Decision Support Systems* **66**, 170–179.
URL: *https://www.sciencedirect.com/science/article/pii/S0167923614001997*

de Ciencia de Datos, I. I. (2020), 'MACHINE LEARNING Y CIENCIA DE DATOS: CóMO SE RELACIONAN', https://i2ds.org/2020/05/19/machine-learning-y-ciencia-de-datos-como-se-relacionan/.

de los Santos, P. R. (2017), 'Tipos de aprendizaje en Machine Learning: supervisado y no supervisado', https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/.

de los Santos, P. R. (2020), 'PLN con Python: Tokens, stop words y ngrams', https://empresas.blogthinkbig.com/pln-con-python-tokens-stop-words-y-ngrams/.

Dietterich, T. (1995), 'Overfitting and undercomputing in machine learning', *ACM computing surveys (CSUR)* **27**(3), 326–327.

EcuRed (2019), 'N-grama', https://www.ecured.cu/N-grama.

EliteDataScience (2017), 'Overfitting in Machine Learning: What It Is and How to Prevent It', https://elitedatascience.com.

Esuli, A. & Sebastiani, F. (2006), Sentiwordnet: A publicly available lexical resource for opinion mining., *in* 'LREC', Vol. 6, Citeseer, pp. 417–422.

Go, A., Bhayani, R. & Huang, L. (2009), 'Twitter sentiment classification using distant supervision', *CS224N project report, Stanford* **1**(12), 2009.

Goldberg, Y. (2017), 'Neural network methods for natural language processing', *Synthesis lectures on human language technologies* **10**(1), 1–309.

González, L. (2021*a*), 'Introducción a Bias y Varianza', https://aprendeia.com/bias-y-varianza-en-machine-learning/.

González, L. (2021*b*), 'Regresión Logística – Teoría', https://cutt.ly/BbKiQqD.

Gujjar, P. & HR, P. K. (2021), 'Sentiment analysis: Textblob for decision making'.

Hecht-Nielsen, R. (1992), Theory of the backpropagation neural network, *in* 'Neural networks for perception', Elsevier, pp. 65–93.

Heras, J. M. (2020), 'Regresión Lineal: teoría y ejemplos en Python', https://www.iartificial.net/regresion-lineal-con-ejemplos-en-python/.

Holgado, P. S., Martán, M. & Herrero, D. B. (2020), 'Del data-driven al data-feeling: análisis de sentimiento en tiempo real de mensajes en español sobre divulgación científica usando técnicas de aprendizaje automático.', *Disertaciones: Anuario electrónico de estudios en Comunicación Social* **13**(1), 4.

HootSuite, WE ARE SOCIAL Y (2020), 'Digital 2020. global digital overview'.

Hurtado, L. F., Pla, F. & Buscaldi, D. (2015), Elirf-upv en tass 2015: Analisis de sentimientos en twitter., *in* 'TASS@ SEPLN', pp. 75–79.

Ichi-Pro (2020-2021), 'Análisis de sentimiento VADER', https://tinyurl.com/xsrxeuzn.

Janssens, O., Van de Walle, R. & Van Hoecke, S. (2015), A learning based approach for real-time emotion classification of tweets, *in* 'Applications of Social Media and Social Network Analysis', Springer, pp. 125–142.

Joyce, B. & Deng, J. (2017), Sentiment analysis of tweets for the 2016 us presidential election, *in* '2017 IEEE MIT Undergraduate Research Technology Conference (URTC)', IEEE, pp. 1–4.

Kottmann, J., Margulies, B., Ingersoll, G., Drost, I., Kosin, J., Baldridge, J., Goetz, T., Morton, T., Silva, W., Autayeu, A. et al. (2011), 'Apache opennlp', *Online (May 2011), www. opennlp. apache. org* .

Kouloumpis, E., Wilson, T. & Moore, J. (2011), Twitter sentiment analysis: The good the bad and the omg!, *in* 'Proceedings of the International AAAI Conference on Web and Social Media', Vol. 5.

Li, H. (2021), 'Aprendizaje automático. Qué es y por qué es importante', https://www.sas.com/es_es/insights/analytics/machine-learning.html.

Liu, B. (2012), 'Sentiment analysis and opinion mining', *Synthesis lectures on human language technologies* **5**(1), 1–167.

Liu, H. & Setiono, R. (1995), Chi2: Feature selection and discretization of numeric attributes, *in* 'Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence', IEEE, pp. 388–391.

Loper, E. & Bird, S. (2002), 'Nltk: The natural language toolkit', *arXiv preprint cs/0205028* pp. 1–8.

Loyola-González, O., Monroy, Raúl y Rodríguez, J., López-Cuevas, A. & Mata-Sánchez, J. I. (2019), 'Clasificación basada en patrones de contraste para la detección de bots en twitter', *IEEE Access* pp. 45800–45817.

Manning, C. D., Raghavan, P. & Schütze, H. (2009), 'Tokenization', https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S. & McClosky, D. (2014), The stanford corenlp natural language processing toolkit, *in* 'Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations', pp. 55–60.

Martin-Domingo, L., Martín, J. C. & Mandsberg, G. (2019), 'Social media as a resource for sentiment analysis of Airport Service Quality (ASQ)', *Journal of Air Transport Management* **78**, 106–115.

Mayo, M. (2018), 'Preprocesamiento de datos de texto: un tutorial en Python', https://medium.com/datos-y-ciencia/preprocesamiento-de-datos-de-texto-un-tutorial-en-python-5db5620f1767.

McCallum, A. K. (2002), 'Mallet: A machine learning for language toolkit', *http://mallet. cs. umass. edu* .

Misra, D. (2019), 'Mish: A self regularized non-monotonic neural activation function', *arXiv preprint arXiv:1908.08681* **4**.

Monkey-Learn (2020), 'Try out our free sentiment analyzer tool with superhuman accuracy', https://monkeylearn.com/sentiment-analysis/.

Muischnek, K. & Mü@ ü risep, K. (2018), corpus de tweets letones e investigación del análisis de sentimientos para letón, *in* 'Human Language Technologies - The Baltic Perspective: Proceedings of the Eighth International Conference Baltic HLT 2018', p. 112.

NetApp (2021), '¿Qué es el aprendizaje profundo?', https://www.netapp.com/es/artificial-intelligence/what-is-deep-learning/#:~:text=A%20diferencia%20de%20los%20algoritmos,lo%20que%20es%20lo%20mismo%2C.

Nguyen, H., Veluchamy, A., Diop, M. & Iqbal, R. (2018), 'Comparative study of sentiment analysis with product reviews using machine learning and lexicon-based approaches', *SMU Data Science Review* **1**(4), 7.

Ollé, J. (n.d.), 'Qué es y para qué sirve el clustering – un ejemplo de aplicación práctico', https://conceptosclaros.com/que-es-clustering/.

Pauli, P. A. (2019), 'Analisis de sentimiento: comparacion de algoritmos predictivos y metodos utilizando un lexicon español', pp. 1–47.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.

Pennebaker, J. W., Francis, M. E. & Booth, R. J. (2001), 'Linguistic inquiry and word count: Liwc 2001', *Mahway: Lawrence Erlbaum Associates* **71**(2001), 2001.

Powerdata (2020), 'Bigdata', https://www.powerdata.es/big-data.

Ray, P. & Chakrabarti, A. (2017), Twitter sentiment analysis for product review using lexicon method, *in* '2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)', IEEE, pp. 211–216.

Rinfret, J. (2019), 'The Hyperparameter Cheat Sheet', https://medium.com/swlh/the-hyperparameter-cheat-sheet-770f1fed32ff.

RODRíGUEZ, C. A. F. (2018), MéTODO DE PREDICCIóN ELECTORAL A TRAVéS DE MODELO BASADO EN ANÁLISIS DE SENTIMIENTOS EN TWITTER, PhD thesis, PONTIFICIA UNIVERSIDAD CATóLICA DE VALPARAíSO.

Roman, V. (2019), 'Aprendizaje No Supervisado en Machine Learning: Agrupación', https://medium.com/datos-y-ciencia/aprendizaje-no-supervisado-en-machine-learning-agrupaci%C3%B3n-bb8f25813edc.

Sabariah, M. K., Effendy, V. et al. (2015), Sentiment analysis on Twitter using the combination of lexicon-based and support vector machine for assessing the performance of a television program, *in* '2015 3rd International Conference on Information and Communication Technology (ICoICT)', IEEE, pp. 386–390.

Sanmartín, F. E. L. (2020), 'Facultad de ingeniería carrera de ingeniería de sistemas'.

SAURA, J. R., Reyes-Menéndez, A. & PALOS-SANCHEZ, P. (2018), 'Un análisis de sentimiento en twitter con machine learning: Identificando el sentimiento sobre las ofertas de# blackfriday', *Revista Espacios* **39**(42).

Singh, V. K. (2016), 'Proposing solution to xor problem using minimum configuration mlp', *Procedia Computer Science* **85**, 263–270.

Sobrino Sande, J. C. (2018), 'Análisis de sentimientos en twitter', pp. 19–22.

Soto, M. G. (2018), 'Episodio 1: Procesamiento de lenguaje natural', https://planetachatbot.com/1-procesamiento-de-lenguaje-natural-1443ff471ed0.

Sproģis, U. & Rikters, M. (2020), 'What can we learn from almost a decade of food tweets', *arXiv preprint arXiv:2007.05194* pp. 1–8.

Stone, P. J., D. D. C. & Smith, M. S. (1966), 'The general inquirer: A computer approach to content analysis'.

Taneja, S., Gupta, C., Goyal, K. & Gureja, D. (2014), An enhanced k-nearest neighbor algorithm using information gain and clustering, *in* '2014 Fourth International Conference on Advanced Computing Communication Technologies', pp. 325–329.

Twitter (2020), 'Twitter usage statistics - internet live stats', https://www.internetlivestats.com/twitter-statistics/. (Accessed on 05/25/2021).

Universidad de Oviedo (2021), 'El algoritmo k-means aplicado a clasificación y procesamiento de imágenes', https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html.

Wan, Y. & Gao, Q. (2015), Un sistema de clasificación de sentimiento conjunto de datos de twitter para el análisis de servicios de aerolíneas, *in* '2015 IEEE international conference on data mining workshop (ICDMW)', IEEE, pp. 1318–1325.

Webb, G. I. (2010), 'Naïve bayes.', *Encyclopedia of machine learning* **15**, 713–714.

Wilson, T., Hoffmann, P., Somasundaran, S., Kessler, J., Wiebe, J., Choi, Y., Cardie, C., Riloff, E. & Patwardhan, S. (2005), Opinionfinder: A system for subjectivity analysis, *in* 'Proceedings of HLT/EMNLP 2005 Interactive Demonstrations', pp. 34–35.

Wu, X., Yang, J. & Wang, S. (2018), 'Tea category identification based on optimal wavelet entropy and weighted k-nearest neighbors algorithm', *Multimedia Tools and Applications* **77**(3), 3745–3759.

xgboost developers (2020), 'XGBoost Documentation', https://xgboost.readthedocs.io/en/latest/.

Zeng, Y. & Zhang, J. (2020), 'A machine learning model for detecting invasive ductal carcinoma with google cloud automl vision', *Computers in Biology and Medicine* **122**, 103861.

Årup Nielsen, F. (2011), 'A new anew: Evaluation of a word list for sentiment analysis in microblogs'.