



Universidad Politécnica  
de Madrid



**Escuela Técnica Superior de  
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Sistema para Determinar si un Mensaje  
Corresponde a un Ataque *Phishing* o es  
*Spam***

Autor: Alejandro Vázquez Gómez

Tutor(a): María del Socorro Bernardos Galindo

Madrid, mayo de 2021

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*

*Grado en Ingeniería Informática*

*Título: Sistema para Determinar si un Mensaje Corresponde a un Ataque  
Phishing o es Spam*

junio de 2021

*Autor:* Alejandro Vázquez Gómez

*Tutor:*

María del Socorro Bernardos Galindo  
Lenguajes y Sistemas Informáticos e Ingeniería de Software  
ETSI Informáticos  
Universidad Politécnica de Madrid

# **Resumen**

Este Trabajo de Fin de Grado describe la construcción de un sistema de clasificación de correos entre aquellos que se puedan catalogar como peligrosos por tener contenido de tipo *phishing* o ser *spam* y aquellos que no lo son.

Para ello, en este proyecto se ha empleado un conjunto de algoritmos de *machine learning*, (en concreto arboles de decisión, k-vecinos más próximos, Bayes ingenuo, y bosques aleatorios). Después se han comparado los resultados de la aplicación de estos algoritmos, resultando el modelo multinomial obtenido a partir de Bayes ingenuo el mejor.

Para ello, se han creado dos modelos de clasificación: uno para correo sospechoso de ser de tipo *spam* o *ham*, y otro que se activará una vez se asegure que dicho correo tiene una URL adjunta, para analizar si lleva o no una web sospechosa de utilizarse para hacer *phishing*. Para ello, se ha usado una serie de conjuntos de datos o *datasets*, de los que se ha utilizado una parte como datos de entrenamiento, y otra parte, como datos de prueba

Además, se ha construido una interfaz para poder obtener los correos de una cuenta de *Gmail* creada específicamente para este proyecto, para poder analizarlos y observar si el sistema completo funciona bien con casos reales de mensajes de correo que pueden llegar en situaciones cotidianas.

# **Abstract**

This Final Degree Project describes the construction of a system for classifying emails between those that can be classified as dangerous for having *phishing*-type content or being *spam* and those that are not.

To do this, this project has used a set of *machine learning* algorithms (specifically decision trees, k-closest neighbors, naive Bayes, and random forests). Afterwards, the results of the application of these algorithms have been compared, with the multinomial model obtained from naive Bayes being the best.

For this, two classification models have been created: one for mail suspected of being *spam* or *ham*, and another that will be activated once it is ensured that said mail has an attached URL, to analyze whether or not it carries a suspicious website used for *phishing*. For this, a series of *datasets* or datasets has been used, part of which has been used as training data, and another part, as test data.

In addition, an interface has been built to obtain the emails from a *Gmail* account created specifically for this project, to be able to analyze them and see if the complete system works well with real cases of emails that can arrive in everyday situations.

# Tabla de contenido

<b>1</b>	<b>Introducción.....</b>	<b>1</b>
1.1	Objetivos .....	1
1.2	Organización del resto de la memoria .....	2
<b>2</b>	<b>Marco teórico .....</b>	<b>2</b>
2.1	<i>Phishing</i> .....	2
2.1.1	Definición de <i>phishing</i> [1] [3] .....	2
2.1.2	Historia del <i>phishing</i> [1] [4].....	3
2.1.3	Funcionamiento de un ataque <i>phishing</i> [3] [4] .....	4
2.1.4	Tipos de técnica <i>phishing</i> [3] [6].....	4
2.1.4.1	<i>Email phishing</i> [6] .....	4
2.1.4.2	<i>Spear phishing</i> [6] [7] .....	5
2.1.4.3	<i>Whaling</i> [6].....	5
2.1.4.4	<i>Smishing y vishing</i> [6] .....	5
2.1.4.5	<i>Angler phishing</i> [6] .....	6
2.2	<i>Spam</i> [2] .....	6
2.2.1	Definición de <i>spam</i> .....	6
2.2.2	Historia del término <i>spam</i> .....	7
2.2.3	Tipos de <i>spam</i> .....	7
2.2.4	<i>spam vs phishing</i> .....	8
2.2.5	<i>Emails ham</i> [11] .....	8
2.3	Bases Teóricas del <i>machine learning</i> [14] [15] .....	9
2.3.1	Aprendizaje supervisado .....	11
2.4	Análisis de métodos a usar en el proyecto .....	12
2.4.1	Modelos lógicos basados en árboles [22] [23] .....	13
2.4.1.1	El espacio de hipótesis.....	13
2.4.1.2	Generalización mínima .....	14
2.4.1.3	Disyunción interna .....	15
2.4.1.4	Caminos por el espacio de hipótesis.....	16
2.4.1.5	Árboles de decisión [24] .....	19
2.4.2	Modelos geométricos basados en distancia [25].....	24
2.4.2.1	Algoritmo de k-vecinos más próximos ( <i>KNN</i> ) [26].....	28
2.4.3	Modelos probabilísticos [27] .....	30
2.4.3.1	Algoritmo de Bayes ingenuo [28] .....	31
2.4.4	Modelos ensamblados [30] .....	37
2.4.4.1	Bosques aleatorios [31] .....	38
<b>3</b>	<b>Desarrollo .....</b>	<b>41</b>
3.1	Arquitectura del proyecto .....	41
3.1.1	<i>Datasets</i> .....	43

3.2 Herramientas usadas .....	44
3.2.1 Entorno de trabajo con <i>Anaconda</i> .....	44
3.2.1.1 <i>JupyterLab</i> .....	45
3.2.2 Librerías usadas en el código .....	47
3.3 Implementación del código .....	48
3.3.1 Clasificación de emails y urls .....	48
3.3.1.1 Carga de datos.....	49
3.3.1.2 Preprocesamiento de datos .....	53
3.3.1.3 Extracción de características .....	58
3.3.1.4 Entrenamiento de algoritmos .....	64
3.3.1.5 Comparación de algoritmos.....	65
3.3.2 Análisis de correos de <i>Gmail</i> .....	66
3.3.2.1 Lectura de correos de la bandeja de entrada .....	66
3.3.2.2 Clasificación de los correos leídos .....	69
<b>4 Resultados y conclusiones .....</b>	<b>72</b>
4.1 Análisis de resultados obtenidos .....	72
4.1.1 Árboles de decision .....	73
4.1.2 K-vecinos más próximos .....	76
4.1.3 Bosques aleatorios .....	78
4.1.4 Bayes Ingenuo .....	80
4.2 Conclusiones.....	82
<b>5 Análisis de Impacto .....</b>	<b>83</b>
5.1 Impacto personal.....	83
5.2 Impacto empresarial.....	84
5.3 Impacto social .....	84
5.4 Impacto económico.....	85
5.5 Impacto medioambiental .....	86
5.6 Impacto cultural.....	86
<b>6 Bibliografía .....</b>	<b>87</b>
<b>Anexo I. Planificación .....</b>	<b>93</b>
Enumeración de tareas .....	93
Diagrama de Gantt .....	93
<b>Anexo II. Descripción de la librería Scikit-learn .....</b>	<b>95</b>
Ajustando y prediciendo: bases del estimador .....	95
Transformadores y preprocesadores.....	96
Tuberías: Encadenamiento de preprocesadores y estimadores .....	96
Evaluación de modelos .....	97
Búsqueda de parámetros automática.....	97

# **Lista de figuras**

Figura 1: Correo estafa de un supuesto "príncipe nigeriano" .....	8
Figura 2: Visión general de cómo se usa el machine learning para llevar a cabo una tarea .....	10
Figura 3: Camino de hipótesis del ejemplo de la Tabla 5 .....	17
Figura 4: Diagrama del camino de hipótesis de la Figura 3 convertido en un árbol .....	18
Figura 5: Un árbol de decisión basado en el árbol de la Figura 4, que aprende de los mismos datos.....	20
Figura 6: (Izquierda) Función de impuridad trazada en contra de la probabilidad empírica de la clase positiva. (Derecha) Construcción geométrica para determinar la impuridad de una bifurcación .....	21
Figura 7: Árbol de decisión que ha aprendido de los datos del ejemplo de la Tabla 5.....	23
Figura 8: (Izquierda) Umbral de decisión resultado de usar dos ejemplares con la regla de decisión de la distancia Euclídea. (Derecha) Mismo caso que la figura de la izquierda, pero usando las reglas de decisión de la distancia de Manhattan .....	26
Figura 9: (Izquierda) Regiones de decisión obtenidas usando tres ejemplares y la regla de decisión de distancias Euclídeas. (Derecha) Mismo caso que el anterior, pero usando la regla de decisión de distancias de Manhattan .....	27
Figura 10: (Izquierda) Mosaico de Voronoi para cinco ejemplares. (Centro) Cogiendo los dos ejemplares más cercanos entre ellos se crea una mayor subdivisión en las células de Voronoi. (Derecha) Las sombras indican que ejemplares contribuyen a que células.....	27
Figura 11: Regiones de decisión para un clasificador de (Izquierda) 3 vecinos más próximos. (Centro) 5 vecinos más próximos. (Derecha) 7 vecinos más próximos .....	29
Figura 12: Basado en la Figura 11, se realiza una ponderación de las distancias sobre los datos usando el método de k-vecinos más próximos con (Izquierda) 3 vecinos más próximos. (Centro) 5 vecinos más próximos. (Derecha) 7 vecinos más próximos .....	30
Figura 13: (Izquierda) Curvas ROC producidas por dos clasificadores de Bayes ingenuo. (Derecha) Mismo dominio que el anterior en un data set diferente, con un umbral del modelo multinomial MAP ligeramente mejor .....	34
Figura 14: (Izquierda) Ejemplo de tres estimadores de densidad en 20 puntos muestreados a partir de una distribución normal con media 0 y varianza unitaria. (Derecha) Aquí, los 20 puntos son muestreados uniformemente a partir de [2, -2], y los métodos no paramétricos son más eficientes .....	37
Figura 15: (Izquierda) Un ensamblado de cinco clasificadores lineales básicos construidos con muestras bootstrap con el método bagging, siendo la regla de decisión la mayoría de voto. (Derecha) Si se transforman los votos en probabilidades, se ve como el ensamblado es un modelo grupal: cada instancia del segmento del espacio obtiene una probabilidad ligeramente diferente .....	38
Figura 16: Arquitectura general del proyecto.....	41
Figura 17: Arquitectura para la lectura y clasificación de los correos de Gmail .....	42
Figura 18: Arquitectura de la distribución de los datadets .....	43
Figura 19: Interfaz visual de JupyterLabs con el código de mi proyecto .....	46
Figura 20: Ejemplo de la figura representada en un texto por wordcloud, con las palabras más frecuentes de dicho texto .....	48
Figura 21: Grafico que muestra el balance de datos entre los emails ham (0) y los emails spam (1) del primer dataset de emails de la Tabla 17 .....	51

Figura 22: Grafico que muestra el balance de datos entre los emails ham (0) y los emails spam (1) del primer dataset de emails de la Tabla 18.....	51
Figura 23: Grafico que muestra el balance de datos entre los emails ham (0) y los emails spam (1) del primer dataset de emails de la Tabla 19.....	52
Figura 24: Grafica que muestra el total de datos de urls etiquetados como phishing (bad) o como no phishing (good).....	53
Figura 25: Grafico comparativo que muestra la cantidad de datos usados como ham (0) y como spam (1) de la concatenación de emails .....	54
Figura 26: Grafica que muestra el total de datos de urls etiquetados como phishing (bad) o como no phishing (good) tras eliminar los datos innecesarios y mapear la columna de Etiquetas.....	55
Figura 27: Gráfica con los datos de emails balanceados.....	56
Figura 28: Gráfica con los datos de urls balanceados.....	56
Figura 29: Diagrama de las palabras más usadas en los emails de tipo ham	59
Figura 30: Diagrama de las palabras más usadas en los emails de tipo spam	60
Figura 31: Diagrama de las palabras más usadas en las urls de tipo no phishing .....	61
Figura 32: Diagrama de las palabras más usadas en las urls de tipo phishing .....	62
Figura 33: Matriz TF-IDF resultado de la transformación con los textos de los emails .....	63
Figura 34: Visualización del correo ejemplo puesto en la bandeja de entrada de Gmail cuando lo he abierto .....	71
Figura 35: Matriz de confusión para las predicciones realizadas por el algoritmo de árboles de decisión en los datos de emails .....	73
Figura 36: Matriz de confusión para las predicciones realizadas por el algoritmo de árboles de decisión en los datos de urls.....	75
Figura 37: Matriz de confusión para las predicciones realizadas por el algoritmo de K-vecinos más próximos en los datos de emails.....	76
Figura 38: Matriz de confusión para las predicciones realizadas por el algoritmo de K-vecinos más próximos en los datos de urls.....	77
Figura 39: Matriz de confusión para las predicciones realizadas por el algoritmo de bosques aleatorios en los datos de emails.....	78
Figura 40: Matriz de confusión para las predicciones realizadas por el algoritmo de bosques aleatorios en los datos de urls.....	79
Figura 41: Matriz de confusión para las predicciones realizadas por el algoritmo de Bayes ingenuo en los datos de emails .....	80
Figura 42: Matriz de confusión para las predicciones realizadas por el algoritmo de Bayes ingeuno en los datos de urls.....	81

## **Lista de tablas**

Tabla 1: Ejemplo del funcionamiento de un concepto de aprendizaje de conjunción.....	13
Tabla 2: Algoritmo para encontrar la GM de un conjunto de instancias .....	14
Tabla 3: Ejemplo de lógica negativa.....	15
Tabla 4: Ejemplo de disyunción interna .....	16
Tabla 5: Ejemplo de datos que no son conjuntivamente separables.....	16
Tabla 6: Algoritmo de CrecerArbol (D, F), que hace crecer un árbol de características a partir de unos datos de entrenamiento .....	19
Tabla 7: Ejemplo de cómo calcular la impuridad de unas características .....	23
Tabla 8: Algoritmo de MejorDivisionEnClase(D, F), que encuentra la mejor ramificación para el árbol de decisión .....	24

Tabla 9: Definición de la distancia de Minkowski .....	25
Tabla 10: Definición de la distancia métrica.....	25
Tabla 11: Ejemplo de mosaicos de Voronoi.....	28
Tabla 12: Ejemplo de predicciones usando el modelo de Bayes ingenuo .....	33
Tabla 13: (Izquierda) Dataset pequeño de emails descritos por un vector de conteo. (Derecha) El mismo data set descrito por un vector de bits .....	35
Tabla 14: Ejemplo para entrenar un modelo de Bayes ingenuo .....	36
Tabla 15: Algoritmo Bagging(D, T, A), que entrena un ensamblado de modelos a partir de muestras bootstrap .....	39
Tabla 16: Algoritmo de BosqueAleatorio(D, T, d), que entrena un ensamblado de modelos basados en árboles a partir de una muestras bootstrap y subespacios aleatorios .....	40
Tabla 17: Primer dataset de emails spam/ham .....	49
Tabla 18: Segundo dataset de emails spam/ham .....	50
Tabla 19: Tercer dataset de emails spam/ham .....	50
Tabla 20: Dataset correspondiente a las urls de tipo phishing/legítimo .....	52
Tabla 21: DataFrame que se corresponde a la concatenación de los tres datasets de emails en uno solo.....	54
Tabla 22: Código creado para obtener un balanceamiento de datos en el dataset de emails .....	55
Tabla 23: Código para limpiar los datos de los emails .....	58
Tabla 24: Código para analizar y limpiar las palabras dentro de las urls .....	58
Tabla 25: Ejemplo de transformación de los emails spam con el vector de valores TF-IDF para pasárselo como argumento al modelo de machine learning .....	63
Tabla 26: Ejemplo de transformación de las urls con el vector de valores CountVectorizer para pasárselo como argumento al modelo de machine learning .....	64
Tabla 27: Esbozo del vector de frecuencia de las 10 primeras palabras de las urls creado con CountVectorizer .....	64
Tabla 28: Código correspondiente a la separación de los datos en los datos de entrenamiento y los datos de prueba .....	64
Tabla 29: Ejemplo de construcción del modelo de machine learning con los árboles de decisión, y su posterior entrenamiento .....	65
Tabla 30: Código correspondiente a la predicción que hace el modelo de machine learning creado con nuevos datos (en este caso, los datos de prueba) .....	65
Tabla 31: Tabla analítica de la precisión de los distintos algoritmos con los datos de entrenamiento y de prueba de los emails.....	66
Tabla 32: Tabla analítica de la precisión de los distintos algoritmos con los datos de entrenamiento y de prueba de las urls .....	66
Tabla 33: Código usado para guardar el modelo ganador en un archivo, para poder ser usado por otras clases dentro de mi proyecto .....	66
Tabla 34: Código para conectarse a mi cuenta de Gmail .....	67
Tabla 35: Código correspondiente a la lectura de Gmail y su almacenamiento en el documento csv creado .....	69
Tabla 36: Esbozo de la cabecera documento csv creado con el remitente, asunto, cuerpo, y urls de los mensajes rescatados de mi cuenta de Gmail.....	69
Tabla 37: Código para cargar el modelo de Bayes Ingenuo y predecir los datos de los correos obtenidos de la cuenta de Gmail .....	70
Tabla 38: Código creado para catalogar los emails en phishing, spam, o ham, dependiendo de los resultados obtenidos .....	71
Tabla 39: Visualización en el documento de texto phishing_gmails.txt del correo de la Figura 34 tras ser analizado por el modelo de Bayes ingenuo, y se clasificado como correo de tipo phishing .....	72

Tabla 40: Informe de clasificación del algoritmo de árboles de decisión en emails .....	74
Tabla 41: Informe de clasificación del algoritmo de árboles de decisión en urls .....	75
Tabla 42: Informe de clasificación del algoritmo de K-vecinos más próximos en emails .....	76
Tabla 43: Informe de clasificación del algoritmo de K-vecinos más próximos en urls .....	77
Tabla 44: Informe de clasificación del algoritmo de bosques aleatorios en emails .....	78
Tabla 45: Informe de clasificación del algoritmo de bosques aleatorios en urls .....	79
Tabla 46: Informe de clasificación del algoritmo de Bayes ingenuo en emails	81
Tabla 47: Informe de clasificación del algoritmo de Bayes ingenuo en urls ....	81

# 1 Introducción

Este proyecto se tiene como objetivo aprender cómo funciona el mundo de la inteligencia artificial (IA) relacionada con el aprendizaje automático o *machine learning*, y aplicar todo esto a la clasificación de correos en aquellos que tengan peligro de tener contenido *phishing* o *spam*, y aquellos que no.

Para entender bien lo que se va a realizar en este proyecto, se debe entender que es el *phishing* y que significa la palabra *spam*. El término *phishing* [1] se puede definir como un tipo de ataque malicioso dedicado a engañar a las personas para que estos compartan información confidencial como contraseñas o números de tarjeta de crédito. Como en cualquier otro tipo de ataque, hay varias formas de realizarlo, pero en el caso de los ataques *phishing*, la práctica más común es que el usuario víctima reciba un mensaje de correo electrónico o un mensaje de texto que imita o suplanta la identidad de una persona u organización de confianza, como pueda ser un compañero de trabajo, un banco, o una oficina gubernamental. Este mensaje contiene un texto destinado a asustar al usuario víctima, a través del uso de ingeniería social, para debilitar su juicio e infundirle miedo. A continuación, el mensaje exige a la víctima que vaya a un sitio web y actúe de inmediato o tendrá que afrontar unas consecuencias. Por otro lado, el *spam* [2] es correo digital basura. Dicho de otro modo, son comunicaciones no solicitadas que se envían de forma masiva por Internet o mediante otros sistemas de mensajería electrónica.

El sistema que pretendo desarrollar en este TFG tiene en cuenta esto para, basándome en técnicas de aprendizaje automático, procesar los correos recibidos para poder analizarlos y clasificarlos en aquellos que tengan contenido *phishing* o *spam*, y en aquellos que no. He concreto he elegido cuatro algoritmos, que son: árboles de decisión, K-vecinos más próximos, Bayes ingenuo, y bosques aleatorios; todos ellos, pertenecientes a técnicas del tipo de aprendizaje supervisado. Para poder entrenar estos algoritmos y probar los modelos resultantes, usaré varios *datasets*, que separaré para dividir entre datos de entrenamiento y datos de prueba.

Una vez entrenados y probados, usaré los modelos creados para probarlos con un caso real. La idea es conectar con una cuenta de *Gmail* creada específicamente para este proyecto, de forma que cuando lleguen correos a esta cuenta, pueda clasificarlos.

En los siguientes apartados voy a desglosar los objetivos de este TFG, y la estructura que va a seguir el mismo para el resto de la memoria.

## 1.1 Objetivos

Los objetivos principales del proyecto son los siguientes:

- Investigar y entender cuáles son los algoritmos de aprendizaje automático usados actualmente para analizar textos.
- Determinar cuál de estas técnicas es la más adecuada para solucionar el problema planteado en el trabajo (es decir, qué tipo y qué algoritmo de *machine learning* utilizaremos).

- Desarrollar el sistema utilizando la técnica elegida que nos permita identificar los mensajes sospechosos de contener *phishing* o *spam*.
- Aplicar el sistema desarrollado a un caso real de correos en línea, como pueda ser *Gmail* o *Outlook*.

## 1.2 Organización del resto de la memoria

El esquema básico de la memoria final va a ser el siguiente (sin incluir los índices y la introducción):

- Marco teórico, donde se explica qué es el *phishing* y *spam*, y en qué consiste el aprendizaje automático y los algoritmos que se han empleado en este trabajo.
- Desarrollo, que contiene la descripción del sistema que se ha creado en este trabajo: la arquitectura general, las herramientas a las que se ha recurrido y cómo se ha llevado a cabo la implementación y las pruebas.
- Resultados y conclusiones, que incluye el análisis comparativo de los distintos modelos construidos y las conclusiones a las que se ha llegado.
- Análisis de impacto, donde se detallan las repercusiones del trabajo.
- Bibliografía, con las publicaciones utilizadas en el estudio y desarrollo del trabajo.
- Anexos

## 2 Marco teórico

En esta sección se va a explicar toda la teoría relacionada con el proyecto, desde explicar qué es el *phishing* hasta una explicación en detalle de los distintos algoritmos de *machine learning* usados en el proyecto.

### 2.1 Phishing

En los siguientes apartados vamos a analizar qué significan los ataques de tipo *phishing*, su historia, sus tipos, y cómo funcionan.

#### 2.1.1 Definición de *phishing* [1] [3]

El *phishing* es el delito de engañar a las personas para que comparten información confidencial y frágil, como puedan ser contraseñas o el número de la tarjeta de crédito. Siendo la palabra *phishing*, pesca, en inglés, este concepto funciona de alguna forma igual que pescar a peces en el mar, y de igual forma, existe más de una forma de atrapar a una víctima, pero hay una técnica muy común, que es enviar a la víctima un correo electrónico o un mensaje de texto que imita (o dicho de otro modo, suplanta la identidad) a una persona u organización de confianza, como un compañero de trabajo, un banco, o una oficina gubernamental. Una vez la víctima recibe el correo o mensaje de texto, y lo abre, se encuentra normalmente con un mensaje pensado para asustarle, siendo la intención debilitar su buen juicio para infundirle miedo. El mensaje recibido suele indicar al usuario víctima que tiene que ir al sitio web y actuar de inmediato, o afrontar las consecuencias negativas.

Si el usuario pica el anzuelo y hace clic en el enlace, se le envía a un sitio web, siendo este una imitación de la página web legítima. Es entonces cuando se le pide al usuario que se registre con sus credenciales de nombre de usuario y de la contraseña. Si el usuario víctima es muy ingenuo e introduce estas credenciales, esta información llega al atacante, y puede robarle la identidad a

la víctima, saquear su cuenta bancaria, o vender esta información en el mercado negro.

A diferencia de otros tipos de amenaza en Internet, el tipo de ataque *phishing* no requiere de conocimientos técnicos especialmente sofisticados. Los atacantes, al usar *phishing*, no tratan de explotar una vulnerabilidad técnica en el sistema, si no utilizar ingeniería social, siendo este concepto explicado como una forma de aprovecharse de la psique humana, a través de su parte emocional, como puede ser el miedo, urgencia, curiosidad, o simpatía, entre otros. Básicamente, es un ataque basado en el factor humano, puesto que muchas veces el eslabón más débil en un sistema de seguridad no es un fallo oculto en el código informático, si no una persona que no comprueba la procedencia de un correo electrónico.

### **2.1.2 Historia del *phishing* [1] [4]**

El término *phishing* viene, en parte, de la palabra anglosajona *fish* (pescar), ya que sus creadores al final lo interpretaron como que este tipo de ataque es una forma de pescar nombres de usuario, contraseñas, u otro tipo de información sensible, en lo que es el mar del Internet; pero también tiene parte de su origen en que en vez de usar la letra “f” se usa “ph” por el término acuñados *phreaks*. Este término viene del famoso hacker *John Draper*, el creador de la *BlueBox* [5], un aparato electrónico que emite diversos tonos por la línea telefónica y permitía realizar pirateo telefónico. Luego el término *phreaking* a la exploración, experimentación, y estudio de los sistemas de telecomunicación.

El *phishing* tiene su origen en los 90, en la compañía *AOL*, una de las compañías proveedoras de servicios de internet por la época, y a la estaban suscritos millones de clientes. La gran popularidad de esta compañía llamó la atención de muchos hackers, y mucha gente que comerciaba con *software* y herramientas piratas e ilegales, empezó a usar *AOL* para sus comunicaciones. De esta forma, crearon un grupo llamado la comunidad *warez*, estableciendo las raíces de lo que luego se conocería como *phishing*.

Inicialmente, los miembros de la comunidad *warez* empezaron robando detalles de usuarios, incluyendo su nombre de usuario, contraseña, y otra información personal. Usando esta información robada junto a un algoritmo que desarrollaron, empezaron a generar número de tarjetas de crédito aleatorias.

Estas tarjetas de crédito luego eran usadas para abrir nuevas cuentas de *AOL*, que se usaban para propósitos nefastos como hacer *spam* a otros usuarios dentro de *AOL*. La compañía *AOL* puso final a esta saga de ataques *phishing* actualizando sus medidas de seguridad.

Habiendo acabado *AOL* con estos ataques *phishing*, los hackers empezaron a buscar otras formas y técnicas de engañar a los usuarios. Empezaron a usar el sistema de mensajería de *AOL* llamado *AIM*, y crearon e-mails falsos personificando a trabajadores de *AOL* para enviar mensajes a clientes dentro del sistema de *AOL*. En estos correos electrónicos falsos, los piratas informáticos pidieron a las personas que verificaran sus cuentas de *AOL* y otra información personal.

Los usuarios inconscientes de este tipo de ataques en *AOL*, cayeron en la trampa. El problema se intensificó cuando los piratas informáticos crearon nuevas cuentas de este sistema de mensajería de *AIM*, y el departamento *TOL* de *AOL* no puede prohibir ni suspender ninguna cuenta creada a través de Internet.

Finalmente, AOL envió mensajes de advertencia y correos electrónicos a sus clientes, pidiéndoles que no revelaran su información personal por correo electrónico y mensajería.

Más adelante, ya entrando en la época de los 2000, con el aumento del comercio electrónico, los piratas informáticos comenzaron a centrar su atención en los clientes de comercio electrónico y los sistemas de pago en línea.

Aunque el intento de conseguir esto no tuvo éxito, el primer ataque de *phishing* conocido en sitios web de comercio electrónico comenzó con el sitio web *E-Gold* en junio de 2001. En 2003, los piratas informáticos registraron varios dominios nuevos que se parecían a nombres de sitios populares como *eBay* y *PayPal*. Luego, utilizando algún software de gusano ilícito, enviaron correos electrónicos falsos a los clientes de *eBay* y *PayPal*. Estos clientes que fueron víctimas de estos correos electrónicos de *phishing* fueron engañados para que proporcionaran los detalles de su tarjeta de crédito y otra información personal.

A principios de 2004, el *phishing* se convirtió en un negocio rentable y los piratas informáticos comenzaron a atacar bancos, empresas y a sus clientes. Una de las principales armas utilizadas por los piratas informáticos durante ese tiempo fue el uso de ventanas emergentes para recopilar información confidencial de usuarios desprevenidos. A partir de ahí, los piratas informáticos comenzaron a idear varias técnicas, que veremos más adelante.

### **2.1.3 Funcionamiento de un ataque *phishing* [3] [4]**

La idea básica detrás de los ataques *phishing* es muy simple. Los adversarios o atacantes se aprovechan de las vulnerabilidades del *software* y la seguridad del empleado y la red de una organización. Estos atacantes crean e-mails especificando que la cuenta de la o las víctimas ha sido suspendida y deben volver a iniciar sesión en cierto sitio que ponen en una *url* para que puedan reactivar su cuenta, y de esta forma los atacantes se hacen con el nombre de usuario y la contraseña del usuario víctima.

A pesar de que mucha gente no lo piensa así estos ataques de tipo *phishing* tienen una planificación previa muy detallada, y realizan la ejecución del ataque de forma muy meticulosa. Los atacantes llevan a cabo el ataque en varias etapas, que incluyen la planificación, la configuración, el asalto, la recolección, y el robo de identidad. Tras el ataque, los cibercriminales evalúan los éxitos y los fallos del ataque, y tras esto, realizan una llamada para organizar el siguiente ataque y como mejorar sus técnicas.

Los usuarios se dan cuenta de que han sido hackeados solo cuando ven que su cuenta está bloqueada, o después de que los atacantes hayan sacado dinero de la cuenta del banco del usuario víctima.

### **2.1.4 Tipos de técnica *phishing* [3] [6]**

Dentro de estos ataques de tipo *phishing*, podemos encontrar varios tipos de estos ataques, y los más conocidos y usados de estos son los que se indican y explican a continuación.

#### **2.1.4.1 Email *phishing* [6]**

La mayoría de los ataques *phishing* son enviados a través de emails. El ladrón registra un dominio falso que imita una presunta organización y envía miles de peticiones genéricas.

Este falso dominio siempre tiene en algún punto una sustitución de caracteres, como usar una “r” y una “n” juntas, de forma que en vez de “m”, usa “rn” para confundir al usuario que ha recibido el correo.

Alternativamente, también es posible que usen el nombre de la organización en la parte del arroba de la dirección email (como *amazon@registrodedominio.com*) con la esperanza de que en la parte del remitente del email que ha enviado el atacante aparezca simplemente “Amazon”, en el buzón de entrada del usuario víctima.

#### **2.1.4.2 Spear phishing [6] [7]**

Este tipo de ataque phishing también está relacionado con el envío de emails. El *spear phishing* se crea para persona específicas en lugar de hacerlo para un grupo amplio de personas. De esta forma, los atacantes pueden personalizar sus comunicaciones y parecer más auténticos. Este *spear phishing* es a menudo el primer paso que se utiliza para penetrar las defensas de una empresa y llevar a cabo un ataque dirigido. Los criminales que realizan este tipo de ataques *phishing* siempre tienen todos o algunos de los siguientes datos de la víctima para poder atacar: su nombre, tu lugar de trabajo, su título de trabajo, su correo de email, o información específica acerca de su trabajo en la organización.

Uno de los ataques más famosos y conocidos de este tipo fue el del hackeo realizado al Comité Nacional Democrático [8]. En este ataque se enviaron emails que contenían contenido malicioso adjunto a más de 1,000 correos electrónicos. Su éxito llevó a lanzar otra campaña que engañaba a los miembros del comité para que compartieran sus contraseñas.

#### **2.1.4.3 Whaling [6]**

Este tipo de ataques *whaling* son más específicos aún. En este los atacantes persiguen llegar hasta un pez gordo dentro de una empresa, como un *CEO*. Estos atacantes suelen dedicar un tiempo considerable a perfilar el objetivo para encontrar el momento oportuno y los medios para robar credenciales de inicio de sesión. Esta *caza de ballenas* es motivo especial de preocupación, puesto que los objetivos de alto nivel a los que se busca llegar, tienen acceso a una gran cantidad de información confidencial de la empresa.

Engaños sencillos como pueden ser *urls* o links falsos no suelen ser muy útiles en este tipo de ataques, puesto que los atacantes están intentando dirigirse a gente senior que tiene más experiencia y está más y mejor protegida frente a ataques de este tipo. De esta forma, en este tipo de ataques son cada vez más recurrentes el uso de impuestos falsos, puesto que los impuestos contienen mucha información valiosa para el o los atacantes que puede usar, como puede ser el nombre de la víctima, su dirección, su número de seguridad social, o información sobre su cuenta bancaria.

#### **2.1.4.4 Smishing y vishing [6]**

Tanto los llamados ataques *smishing* como *vishing*, hacen uso de los teléfonos, que reemplazan al correo electrónico como método de comunicación. Por un lado, el *smishing* implica que los delincuentes envíen mensajes de texto (cuyo contenido es muy similar al del *phishing* por correo electrónico), y por otro lado el *vishing* implica engañar a una víctima elegida por el o los atacantes a través de una conversación telefónica.

Una estafa de *vishing* común involucra a un criminal que se hace pasar por un investigador de fraude (ya sea de la compañía de la tarjeta o del banco) que le dice a la víctima que su cuenta bancaria ha sido de alguna forma violada.

A continuación, el delincuente le pedirá a la víctima que proporcione los datos de la tarjeta de pago para verificar su identidad, o para transferir dinero a una cuenta "segura", es decir, la cuenta del delincuente.

#### **2.1.4.5 Angler phishing [6]**

Este es un vector de ataque relativamente nuevo; se basa en el engaño a través de las redes sociales, en las cuales se ofrecen una serie de formas para que los delincuentes engañen a las personas. *Urls* falsas, sitios web, publicaciones y *tweets* clonados, o la mensajería instantánea (que es en esencia similar al *smishing*) se pueden utilizar para persuadir a las personas de que divulguen información confidencial o descarguen malware.

Alternativamente, los delincuentes pueden usar los datos que las personas publican voluntariamente en las redes sociales para crear ataques altamente dirigidos.

Uno de los ataques más famosos de *angler phishing* fue realizado en 2016, en el cuál mil de usuarios de *Facebook* recibieron mensajes en los que se les decía que habían sido mencionados en una publicación. El mensaje había sido iniciado por delincuentes y desató un ataque en dos etapas. La primera de ambas etapas descargó un troyano que contenía una extensión maliciosa del navegador *Chrome* en la computadora del usuario.

En la siguiente etapa, una vez que el usuario iniciaba sesión en *Facebook* utilizando el navegador comprometido, el delincuente podía secuestrar la cuenta del usuario. De esta forma, pudieron cambiar la configuración de privacidad, robar datos y propagar la infección a través de los amigos de *Facebook* de la víctima.

## **2.2 Spam [2]**

En los siguientes apartados vamos a analizar que significa el famoso término *spam*, su historia, y cómo funcionan los mensajes calificados como tal.

### **2.2.1 Definición de *spam***

Cualquier usuario que haya estado más de unos pocos segundos en Internet se habrá encontrado con *spam*. Parece una parte inseparable de la experiencia en línea, algo que aceptamos como normal. Pero ¿cómo se puede definir el *spam*? ¿Qué diferencia los correos electrónicos de *spam* de los demás? En términos de Internet, ¿qué significa el *spam*?

La respuesta es que el *spam* nunca es solicitado. Es molesto, normalmente es promocional, se envía a muchísimas personas y llega, aunque no se haya pedido.

Si definimos *spam* como mensajes masivos no solicitados, el *spamming* es el acto de enviar estos mensajes, mientras que a la persona que participa en esta práctica se la denomina *spammer*. La mayoría de las veces, el *spam* es de naturaleza comercial y, aunque es preocupante, no es necesariamente malicioso o fraudulento (aunque puede serlo en ocasiones).

## 2.2.2 Historia del término *spam*

El uso del término *spam* para describir este tipo de mensajes intrusivos masivos es una referencia a un famoso sketch del grupo de comediantes ingleses *Monty Python*. En él, un grupo de comensales (vestidos como vikingos) proclama de forma repetida y vociferante que todo el mundo debe comer *Spam*, tanto si les gusta como si no. De un modo similar, un *spammer* le llena el buzón del correo electrónico con mensajes no deseados.

Cuando se escribe con s mayúscula, como *Spam*, el término se refiere a la carne de cerdo enlatada que tanto gusta a los vikingos antes mencionados. Aquí se utilizará con la letra s minúscula para hablar de la interminable inundación de correos electrónicos y otros mensajes que nadie ha pedido.

## 2.2.3 Tipos de *spam*

Según los *Monty Python*, el *Spam* puede freírse, hornearse, hacerse con huevos revueltos, meterse en un sándwich o incluso servirse con arroz y algas. Cuando hablamos de la variedad electrónica, el menú es igual de diverso. A continuación, se presenta una lista de lo que cabe esperar en el amplio mundo del *spam*:

- **spam por correo electrónico:** Es el *spam* más habitual. Inunda la bandeja de entrada del servicio de mensajería, y distrae al usuario de los mensajes que sí le interesan. Este tipo de *spam* se pueden ignorar completamente si así lo desea el usuario.
- **spam SEO:** También es conocido como *spamdexing* y es el abuso de los métodos de optimización de los motores de búsqueda (SEO) para mejorar la valoración de búsqueda del sitio web del *spammer*. Se puede dividir el *spam* SEO en dos amplias categorías:
  - **spam de contenido:** Los *spammers* llenan sus páginas de palabras clave populares, a menudo no relacionadas con el sitio web, para intentar colocarlo más arriba en las búsquedas de dichas palabras. Otros reescriben el contenido existente para que sus páginas parezcan más sustanciales y exclusivas.
  - **spam de enlaces:** Básicamente, se basa en un comentario de un blog o en una publicación de un foro con enlaces irrelevantes. El *spammer* intenta aprovechar una mecánica de SEO conocida como *backlinking* para atraer tráfico hacia su página. El *blacklinking* [9] es básicamente un enlace externo que conduce a un sitio web o una página web. La calidad de un *backlink* depende de la reputación en línea de este sitio web o página web y de su credibilidad ante los ojos de los buscadores.
- **spam de redes sociales:** A medida que Internet se va haciendo más social, los *spammers* tratan de aprovecharse para propagar su *spam* mediante cuentas falsas de usar y tirar en redes sociales populares como *Facebook* o *Instagram*.
- **spam de móvil:** *spam* en formato SMS. Además de mensajes de texto, algunos *spammers* utilizan también notificaciones *push* [10] (mensajes que se envían de forma directa desde el servidor a dispositivos móviles, como teléfonos inteligentes o tabletas, y aplicaciones de escritorio, en todo tipo de sistemas operativos) para llamar la atención sobre sus ofertas.
- **spam de mensajería:** es como el *spam* por correo electrónico, pero más rápido. Los *spammers* lanzan sus mensajes en plataformas de mensajería instantánea como *WhatsApp*, *Skype* o *Snapchat*.

## 2.2.4 *spam* vs *phishing*

La principal diferencia entre el *spam* y el *phishing* es la intención del responsable. Los *spammers* son una molestia, pero normalmente no buscan hacerle daño a ningún usuario de la red. Tienen algo para vender y han decidido que el *spam* es una técnica eficaz para promocionar su producto, oferta o servicio.

Los *phishers*, por otro lado, son ciberdelincuentes que buscan hacerse con información confidencial, ya sea mediante engaños o con el uso de *malware*. Como el *spam*, las estafas de *phishing* suelen enviarse de forma masiva por correo electrónico, pero con fines más dañinos, como pueden ser el fraude, el robo o incluso el espionaje industrial.

Por poner un ejemplo, el correo electrónico que se muestra en la *Figura 1* es un ejemplo de la estafa de anticipo del «príncipe nigeriano», cuyo objetivo es el robo de dinero.

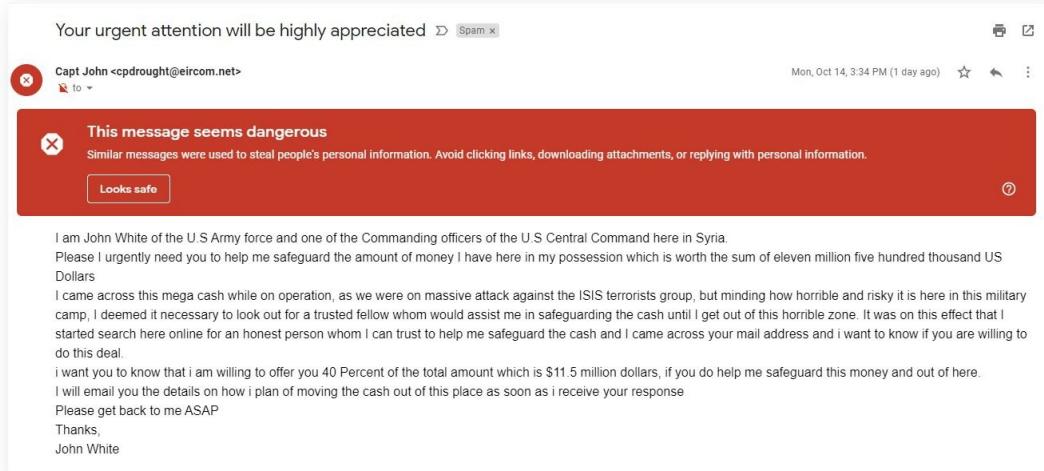


Figura 1: Correo estafa de un supuesto "príncipe nigeriano"

## 2.2.5 Emails *ham* [11]

El término *ham* fue acuñado originalmente por *SpamBayes* [12] alrededor de 2001. Este es un filtro de spam bayesiano escrito en Python que utiliza técnicas expuestas por el ingeniero informático *Paul Graham* en su ensayo *Un plan para el spam*. Actualmente se define y se entiende como "Correo electrónico que generalmente se desea y no se considera spam".

Hay dos formas en las que un usuario de la red podría haberse registrado para recibir este correo electrónico, como es:

- **Directamente:** mientras descargaba software gratuito, como un navegador o un juego, o se registraba en un nuevo servicio en línea, se le pedía que aceptara y marcarla la casilla de aceptación de sus Términos de servicio. Por debajo o por encima de estos términos de servicio suele haber otras casillas de verificación. El usuario podría haber hecho click en la respuesta afirmativa sin fijarse a viene ligada a recibir información

y ofertas para el usuario y sus socios, y en tal caso, solicitó legalmente este tipo de correo electrónico.

- **Indirectamente:** este es el mismo escenario que el registro directo, excepto que la casilla de información y ofertas está previamente marcada, lo que le permite desmarcar la casilla si no desea estar en sus listas de correo.

De cualquier manera, una vez que esté en una lista de correo masivo, se pueden enviar legalmente las ofertas (y rara vez cualquier información que valga la pena) siempre que cumplan con las regulaciones de *RFC (Request for Comments)* [13], siendo estas una serie de publicaciones del grupo de trabajo de ingeniería de internet que describen diversos aspectos del funcionamiento de Internet y otras redes de computadoras, como protocolos, procedimientos, etc..., y comentarios e ideas sobre estos.

La buena noticia es que, si se siguen las reglas del *RFC*, es fácil detener estos correos electrónicos. Todo lo que se tiene que hacer es simplemente hacer *click* para cancelar la suscripción y este tipo de correos masivos se detiene, siempre y cuando entren del marco legal.

Los *spammers* maliciosos se aprovechan especialmente de esto y ofrecen el mismo formato en la parte inferior de sus correos electrónicos, vinculando el enlace de cancelación de suscripción a descargas maliciosas y/o *cookies* de seguimiento entre otros.

Dicho esto, una vez discutido que son, cómo funcionan, y que tipos de *phishing* y *spam* hay, ¿cómo solucionamos el no caer en esta trampa de mensaje *phishing* o *spam*? Existen muchas formas de combatirlo, humanas o con aplicaciones que actúan una vez detectada la trampa. Pero la forma de actuar de nuestro proyecto va a ser un sistema previsor que permitirá detectar los emails que contienen contenido *phishing* y/o *spam*, y clasificarlos. Esta es una de las soluciones más rápidas y eficientes, y para ellos vamos a usar algoritmos de *machine learning*, para detectar mensajes que puedan ser sospechosos de tener este tipo de contenidos engañosos. Pero, ¿Cuáles son las bases teóricas del *machine learning*? En el capítulo siguiente se va a explicar en detalle.

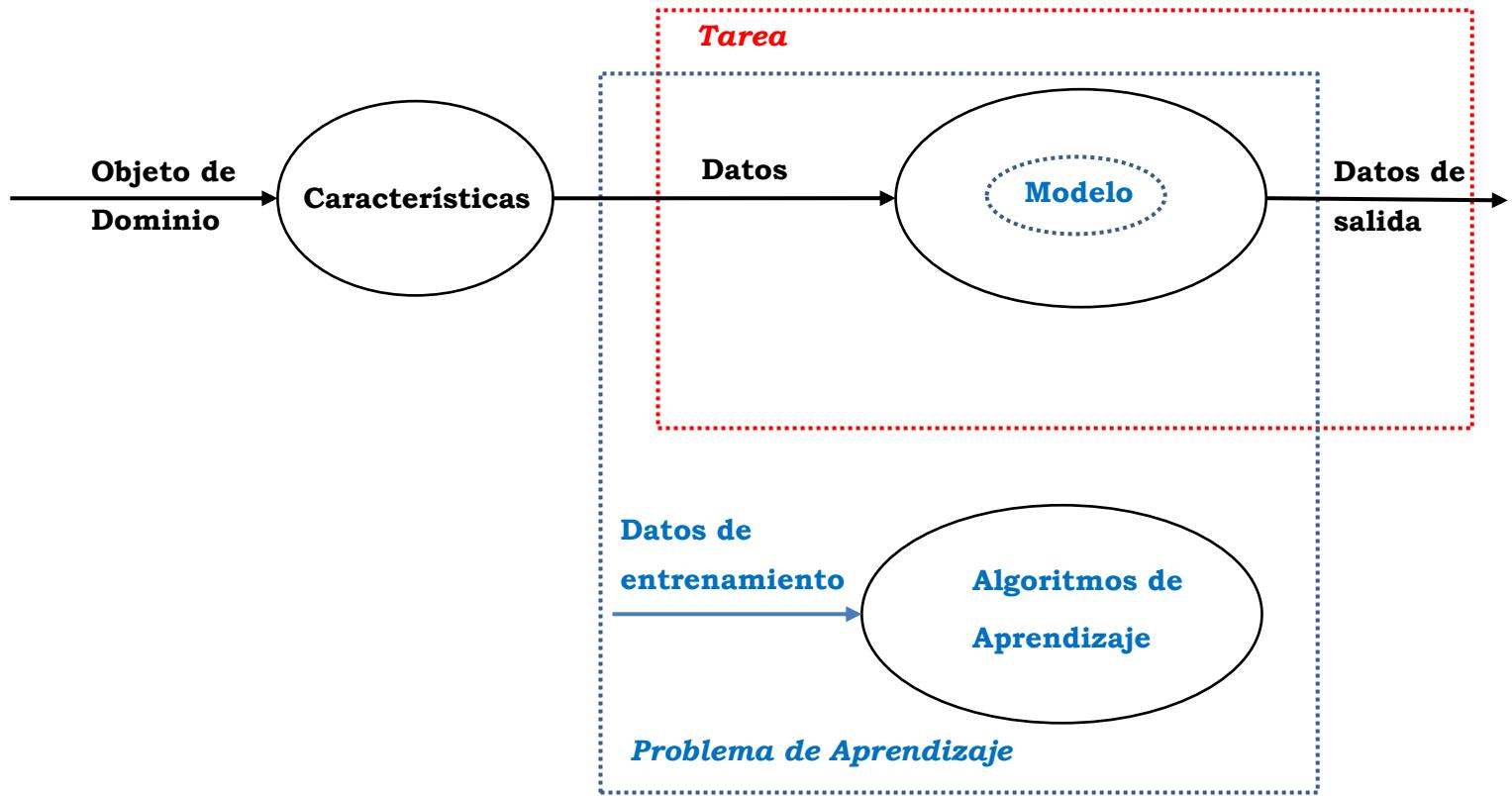
## 2.3 Bases Teóricas del *machine learning* [14] [15]

Hoy en día hay cada vez se confía más y más en el uso de modelos algorítmicos para resolver problemas complejos, teniendo un tremendo éxito la modelización de inteligencia biológica y natural, que dan lugar a los llamados “sistemas inteligentes”. Junto a la lógica, razonamiento deductivo, sistemas expertos, razonamiento basado en casos, y los sistemas de *machine learning*, estos sistemas de inteligencia forman parte del campo de la Inteligencia Artificial. Al final, el *machine learning* no es más que una rama más de la *IA* que permite que las máquinas aprendan sin ser expresamente programadas para ello. Estos sistemas de *machine learning* se caracterizan principalmente por identificar patrones entre unos datos dados para hacer predicciones.

Actualmente estas tecnologías se usan para cientos de aplicaciones como pueden ser las recomendaciones de *Netflix* o *Spotify*, las respuestas inteligentes de *Gmail*, o la capacidad de poder conversar con *Siri* en nuestros móviles, o *Alexa* de *Amazon*.

En esta sección voy a dar unas pinceladas sobre que es el *machine learning* dentro del campo de la *IA*, y todos los posibles métodos de aprendizaje básicos (aprendizaje supervisado, de refuerzo, y no supervisado). Luego, voy a

contemplar cuál es el aprendizaje que más conviene usar, y dentro de ese aprendizaje, el mejor método o algoritmos de entre varios que analizaremos.



*Figura 2: Visión general de cómo se usa el machine learning para llevar a cabo una tarea*

El *machine learning* básicamente es el uso de unas características determinadas para construir unos modelos que consigan alcanzar unos objetivos y cumplimentar unas tareas que queremos suplir, básicamente como se muestra en el esquema de la *Figura 2*.

En esencia, las características definen un “lenguaje” en el que se describen los objetos más relevantes del dominio, pudiendo estos ser desde emails hasta complejas moléculas orgánicas. Normalmente, no se debería volver a hacer uso de los objetos de dominio una vez se tenga una representación de las características; razón por la cual estas características juegan un papel tan importante en el *machine learning*. Por otro lado, una *tarea* es la representación abstracta de un problema que se quiere resolver teniendo en cuenta estos objetos de dominio, siendo la forma más concurrida clasificar los objetos en dos o más clases. Muchas de estas tareas pueden ser representadas como un mapeo que transforma puntos de datos en unos datos de salida claros. Este mapeo o modelo es producido por un algoritmo de *machine learning* aplicado a los datos de entrenamiento. Hay muchos de estos modelos, pero más adelante se verá que en mi caso, para el problema de este proyecto, solo me interesan algunos particulares.

Por otro lado, también interesa a veces abandonar las nociones de clases discretas de clasificación, y predecir número reales. A esta tarea se le llama *regresión*, y básicamente consiste en aprender una función de valor real a partir de casos de entrenamiento etiquetados como funciones con valores verdaderos.

Teniendo en cuenta todo esto, se puede catalogar mi sistema basándonos en si se tienen o no unos datos de antemano, entre otros factores, a la hora de completar una tarea dentro de un sistema de *machine learning*. De esta forma, se pueden encontrar tres tipos de aprendizaje, que se van a explicar brevemente a continuación [16]:

- **Aprendizaje supervisado:** en este tipo de aprendizaje se provee al modelo con un data set consistente de vectores de entrada y un objetivo (la salida deseada) asociado con cada vector de entrada. Se refiere a este conjunto de datos o data set como el set de entrenamiento.
- **Aprendizaje no supervisado:** en este tipo de aprendizaje el objetivo es descubrir patrones o características en los datos de entrada sin ninguna asistencia de parte de un agente externo. Muchos de los métodos de aprendizaje no supervisado realizan un agrupamiento de los patrones surgidos en el proceso de aprendizaje.
- **Aprendizaje de refuerzo:** en este tipo de aprendizaje, el objetivo es recompensar de alguna forma al modelo por tener un buen rendimiento, y penalizarlo en caso de que el rendimiento haya sido negativo.

Sabiendo esto, ¿cuál es el mejor método de aprendizaje para mi proyecto? Y dentro de ese método de aprendizaje, ¿Qué algoritmo me interesa más usar? En mi caso he elegido el aprendizaje supervisado, del cual voy a explicar mi razonamiento de elección en la siguiente subsección.

### 2.3.1 Aprendizaje supervisado

Como ya se ha explicado en anteriores apartados, el proyecto que pretendo desarrollar se basa en entrenar a un sistema de *machine learning* con *datasets* de emails como los objetos de dominio, y una vez el algoritmo de aprendizaje creado haya entrenado con estos *datasets*, utilizando unos modelos o algoritmos concretos, se pretende que este modelo, cogiendo emails de mi *Gmail*, puede analizar y clasificar si corresponde o no a un email con un contenido propio de un email *phishing*.

Sabiendo esto, se debe analizar en las siguientes secciones el tipo de aprendizaje que necesitamos, así como dentro del tipo de aprendizaje escogido, cuál es o son los algoritmos más interesantes de usar.

Una vez explicado los distintos métodos de aprendizaje, teniendo en cuenta que el proyecto consiste en la detección de mensajes *phishing*, se puede descartar el método de aprendizaje por refuerzo, ya que basar mi sistema en un aprendizaje de prueba-error no tendría mucho sentido. Entonces, quedaría por elegir entre el aprendizaje supervisado y no supervisado. ¿Cuál sería mejor de estos dos? En el aprendizaje no supervisado se puede intentar encontrar patrones en los mensajes de texto o correos electrónicos que usaría como *datasets* a través de *clustering*. Esto es aprender patrones de los emails sin indicar ninguna característica concreta, el sistema de *machine learning* debe identificar los patrones por el mismo sin ninguna ayuda. Esta forma de aprendizaje es interesante, pero puede acabar siendo poco eficiente para mi proyecto, puesto que es difícil determinar que correos pueden ser o no de tipo *phishing* buscando patrones en dichos mensajes. Lo mejor es que haya una intervención externa, que entrene al sistema creado para que aprenda que mensajes son sospechosos, y cuáles no, sabiendo cual es el resultado óptimo que se espera a la salida del sistema de capas de aprendizaje. Así pues, voy a optar por un método de aprendizaje de tipo supervisado.

El aprendizaje supervisado [16] requiere de un *dataset* de entrenamiento que consiste de unos vectores de entrada y un vector objetivo asociado con cada vector de entrada. Los modelos que van a aprender de estos datos de entrada usan el vector objetivo para determinar como de bien ha aprendido, así como para orientar los ajustes de los pesos para hacer el sistema lo más óptimo posible, y de esta forma, reducir el error hasta términos óptimos.

Dicho esto, voy a analizar ahora que algoritmos se van a usar dentro del proyecto.

## 2.4 Análisis de métodos a usar en el proyecto

Dentro de los tipos de aprendizajes supervisados hay varios algoritmos y métodos que tienen distintos usos. Existen modelos basados en árboles; modelos basados en reglas; modelos lineares; modelos basados en distancia; modelos estadísticos; y modelos de ensamblado. Pero finalmente hay cuatro de ellos específicamente que me han interesado, ya que son algoritmos usados para clasificar datos de forma eficiente (los correos de nuestros *datasets*), y aunque probablemente haya más, me han llamado la atención en particular estos cuatro, porque además reúnen cinco formas de clasificación bastante distintas que puede ser bastante interesante implementar y analizar. Estos métodos son [17]:

- **Árboles de decisión** [18]: En este algoritmo basado en árboles, los datos de entrada son continuamente separados y clasificados según unos parámetros que se deben indicar. Funciona, como bien dice el nombre, como un árbol, con ramas de decisión como las líneas que dividen los datos; y con hojas, como las decisiones tomadas por el algoritmo para clasificar los datos (es decir, los datos de salida de la red de entrenamiento).
- **K-vecinos más próximos** [19]: Este modelo basado en distancia, clasifica cada dato nuevo en el grupo que le corresponda según tenga  $k$  vecinos más cerca de un grupo o de otro, calculando la distancia del nuevo dato a cada uno de los existentes, y ordenando dichas distancias de menor a mayor para seleccionar el grupo al que pertenece.
- **Bayesiano ingenuo** [20]: Este modelo probabilístico se basa en el teorema de Bayes, el cual encuentra la probabilidad de que ocurra un evento  $A$  ocurra sabiendo que otro evento  $B$  ya ha ocurrido. Su representación matemática es:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

donde  $P(X)$  es la probabilidad de que suceda el evento  $X$  y  $P(X|Y)$  es la probabilidad de que suceda el evento  $Y$  habiendo observado el evento  $X$ .

- **Bosques aleatorios** [21]: Como bien indica su nombre, este modelo se basa en el ensamblado de árboles de decisión individuales que operan como conjuntos. Cada árbol individual en el bosque aleatorio suelta una clase predictora, y la clase con más votos se convierte en nuestro modelo de predicción.

En los siguientes capítulos voy a analizar cada uno de los cinco métodos. Mi objetivo es implementar los algoritmos correspondientes, y ver y analizar cuál es el más eficiente y rápido para el proyecto planteado.

## 2.4.1 Modelos lógicos basados en árboles [22] [23]

Los modelos basados en árboles son los modelos de *machine learning* más populares en la actualidad. Los árboles son expresivos y fáciles de entender, y de una particular atracción por parte de los informáticos debido a su naturaleza recursiva de “divide y conquista”.

Los modelos basados en árboles al final son modelos lógicos, los cuales como bien indica su nombre, implementan expresiones lógicas para dividir el espacio como instancia en segmentos, y de esa forma construir modelos de agrupación. El objetivo es encontrar una segmentación de forma que los datos en cada segmento del espacio sean más homogéneos, en relación a la tarea que se quiere resolver. A la hora de realizar una clasificación el objetivo es encontrar una segmentación de forma que las instancias en cada segmento son predominantemente de una clase específica. Un árbol consiste en un conjunto de implicaciones con reglas *if-then*, donde la parte de las reglas de los *if* define un segmento y está organizado en una estructura de árbol, mientras que la parte de las reglas de los *then* define el comportamiento del modelo en un segmento particular. Para entender mejor cómo funcionan los modelos lógicos, hay que entender qué es el espacio de hipótesis.

### 2.4.1.1 El espacio de hipótesis

El entorno de aprendizaje de conceptos más simple se basa en la restricción de expresiones lógicas que describen de conceptos a conjunciones. Para entender lo que acabamos de decir mejor, debemos observar el ejemplo de la *Tabla 1*.

Suponiendo que se encuentra un conjunto de animales de mar que se sospecha que pertenecen a la misma especie. Se observa su tamaño de longitud en metros, si tienen o no branquias, si tienen un pico prominente, y si tienen pocos o muchos dientes. Usando estas características, se puede describir al primer animal que se encuentra con la siguiente conjunción:

Tamaño = 3  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos

El siguiente animal tiene las mismas características, pero mide un metro más, así que se descarta la condición de tamaño y se generaliza la conjunción como:

Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos

El tercer animal tiene de nuevo 3 metros de longitud, tiene pico, no tiene branquias, y tiene pocos dientes, así que la descripción generalizada quedaría como:

Branquias = no  $\wedge$  Pico = si

El resto de animales cumplen la misma conjunción de características de no tener branquias y tener pico, por lo que finalmente se decide si es algún tipo de delfín.

*Tabla 1: Ejemplo del funcionamiento de un concepto de aprendizaje de conjunción*

A pesar de la simplicidad del ejemplo de la *Tabla 1*, el espacio de posibles conceptos o el espacio de hipótesis, es ya de por sí bastante largo. En el supuesto de que se tienen ahora 3 posibles tamaños de longitud: de 3, 4, y 5 metros, mientras que las otras características siguen teniendo dos valores posibles cada una. Se tienen entonces  $3 \cdot 2 \cdot 2 \cdot 2 = 24$  posibilidades. ¿Cuántos conceptos de conjunción están usando entonces las mismas características? Se

puede responder a esta pregunta si se toman también como una posibilidad de valor la abstención de una característica en la conjunción. Esto da entonces como resultado  $4*3*3*3 = 108$  conceptos diferentes. Aunque esto de por sí ya parezca mucho, hay que darse cuenta que el número de posibles extensiones del espacio de hipótesis es mucho más largo, de  $2^{24}$ , que es más de 16 millones de posibilidades. Esto significa que, si se coge un conjunto aleatorio de instancias, las posibilidades de encontrar un concepto de conjunción que describa exactamente esas instancias son de 100,000 a 1. Esto es bastante positivo, puesto que permite al sistema que aprende, generalizar más allá de los datos de entrenamiento, y cubrir instancias que no haya visto todavía.

#### 2.4.1.2 Generalización mínima

Si se establecen unas reglas para todos los conceptos que no cubren al menos una instancia en el ejemplo de la *Tabla 1*, el espacio de hipótesis se reduce a 32 conceptos de conjunción. Insistiendo en que cualquier hipótesis cubre las tres instancias se reduce el espacio a solo cuatro conceptos, siendo el menos general el que se encuentra al final del ejemplo de la *Tabla 1*, en lo que se llama generalización mínima (*GM*). El algoritmo de la *Tabla 2* formaliza el procedimiento, que se basa en aplicar repetidamente la operación de *GM* (que se explica en el algoritmo de la *Tabla 2*) a una instancia y una hipótesis, puesto que ambas tienen la misma forma lógica. La estructura de la hipótesis se asegura de que el resultado es independiente del orden del orden en el que las instancias son procesadas.

Intuitivamente, la *GM* de dos instancias es el concepto más próximo en el espacio de hipótesis donde los caminos hacia arriba de ambas instancias se intercalan. El hecho de que este punto de intersección es único es una propiedad especial de muchos espacios de hipótesis lógicos, y se puede hacer buen uso de ello en el aprendizaje. Siendo más preciso, tal espacio de hipótesis forma un enrejado: un orden parcial en el cual cada dos elementos tienen un límite mínimo superior (*LMS*) y un límite máximo inferior (*LMI*). Así pues, la *GM* de un conjunto de instancias es justamente el *LMS* de las instancias de ese enrejado. Yendo más allá, es el *LMI* del conjunto de todas las generalizaciones de la instancia: todas las posibles generalizaciones son al menos tan generales como la *GM*. En este sentido, la *GM* es la generalización más conservativa que puede aprender de los datos.

Input: datos  $\mathbf{D}$

Output: Expresión lógica  $\mathbf{H}$

```

1  $x \leftarrow$  primera instancia de  $\mathbf{D}$ 
2  $\mathbf{H} \leftarrow x$ 
3 while {instancias izquierdas} do
4      $x \leftarrow$  Siguiente instancia de  $\mathbf{D}$ ;
5      $\mathbf{H} \leftarrow \mathbf{GM}(\mathbf{H}, x);$ 
6 end
7 return  $\mathbf{H}$ 
```

*Tabla 2: Algoritmo para encontrar la GM de un conjunto de instancias*

Existe la posibilidad de que dentro del ejemplo de la *Tabla 1* se pueda construir un espacio de hipótesis lógicas con sobre generalización, como pueda ser **Branquias = no**. Para evitar esto, se usa la lógica negativa, como se explica en el ejemplo de la *Tabla 3*.

En el ejemplo de la *Tabla 1* se observaban los animales con las siguientes características:

**p1:** Tamaño = 3  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos

**p2:** Tamaño = 4  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos

**p3:** Tamaño = 3  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = pocos

Se imagina ahora un animal que claramente no pertenece a la especie de los tres anteriores casos; es decir, un caso negativo. Se puede describir con la siguiente conjunción:

**n1:** Tamaño = 5  $\wedge$  Branquias = si  $\wedge$  Pico = si  $\wedge$  Dientes = muchos

Este ejemplo negativo excluye algunas de las generalizaciones que hasta ahora eran posibles: en particular, excluye el concepto **Pico = si**, así como los conceptos vacíos que postulaban que todos eran un delfín.

*Tabla 3: Ejemplo de lógica negativa*

#### 2.4.1.3 Disyunción interna

A partir de los ejemplos que se han observado anteriormente es posible que podamos concluir que siempre se tiene una hipótesis única más general, pero no es este el caso en general. Para demostrar esto, se introduce una forma restringida de disyunción llamada disyunción interna. La idea es muy simple: si se observa un delfín que mide 3 metros, y otro que mide 4 metros, se añade la condición “El tamaño es de 3 o 4 metros” al concepto. Se escribe esto como **Tamaño = [3, 4]**, que lógicamente significa **Tamaño = 3 v Tamaño = 4**. Esto, por supuesto, solo tiene sentido para características que tengan más de dos valores: por ejemplo, la disyunción interna **Dientes = [Muchos, Pocos]** siempre es verdad y se puede descartar. En el ejemplo de la *Tabla 4* se puede apreciar un ejemplo de disyunción interna.

Usando los mismos ejemplos positivos del ejemplo de la *Tabla 1*, la segunda y tercera hipótesis son ahora:

**Tamaño = [3, 4]  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos**

y

**Tamaño = [3, 4]  $\wedge$  Branquias = no  $\wedge$  Pico = si**

Se pueden descartar cualesquiera de estas tres condiciones en la última GM sin cubrir los casos negativos del ejemplo de la *Tabla 3*. Si generalizamos más a condiciones simples, se puede ver qué **Tamaño = [3,4]** y **Branquias = no** se pueden considerar correctas, pero **Pico = si** no, ya que cubre el ejemplo negativo.

*Tabla 4: Ejemplo de disyunción interna*

#### 2.4.1.4 Caminos por el espacio de hipótesis

Ahora, ¿qué sucede cuando la GM de ejemplos positivos cubre uno o más negativos? En este caso, cualquier generalización de GM será también inconsistente. Dicho de otra forma, cualquier hipótesis consistente estará incompleta. Se da el caso entonces que el espacio sobrante está vacío, y se dice que los datos en este caso no son conjuntivamente separables. El ejemplo de la *Tabla 5* cubre este caso.

Se imagina ahora que se tienen los siguientes casos positivos:

- p1:** Tamaño = 3  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos
- p2:** Tamaño = 4  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos
- p3:** Tamaño = 3  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = pocos
- p4:** Tamaño = 5  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = muchos
- p5:** Tamaño = 5  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = pocos

Y los siguientes casos negativos:

- n1:** Tamaño = 5  $\wedge$  Branquias = si  $\wedge$  Pico = si  $\wedge$  Dientes = muchos
- n2:** Tamaño = 4  $\wedge$  Branquias = si  $\wedge$  Pico = si  $\wedge$  Dientes = muchos
- n3:** Tamaño = 5  $\wedge$  Branquias = si  $\wedge$  Pico = no  $\wedge$  Dientes = muchos
- n4:** Tamaño = 4  $\wedge$  Branquias = si  $\wedge$  Pico = no  $\wedge$  Dientes = muchos
- n5:** Tamaño = 4  $\wedge$  Branquias = no  $\wedge$  Pico = si  $\wedge$  Dientes = pocos

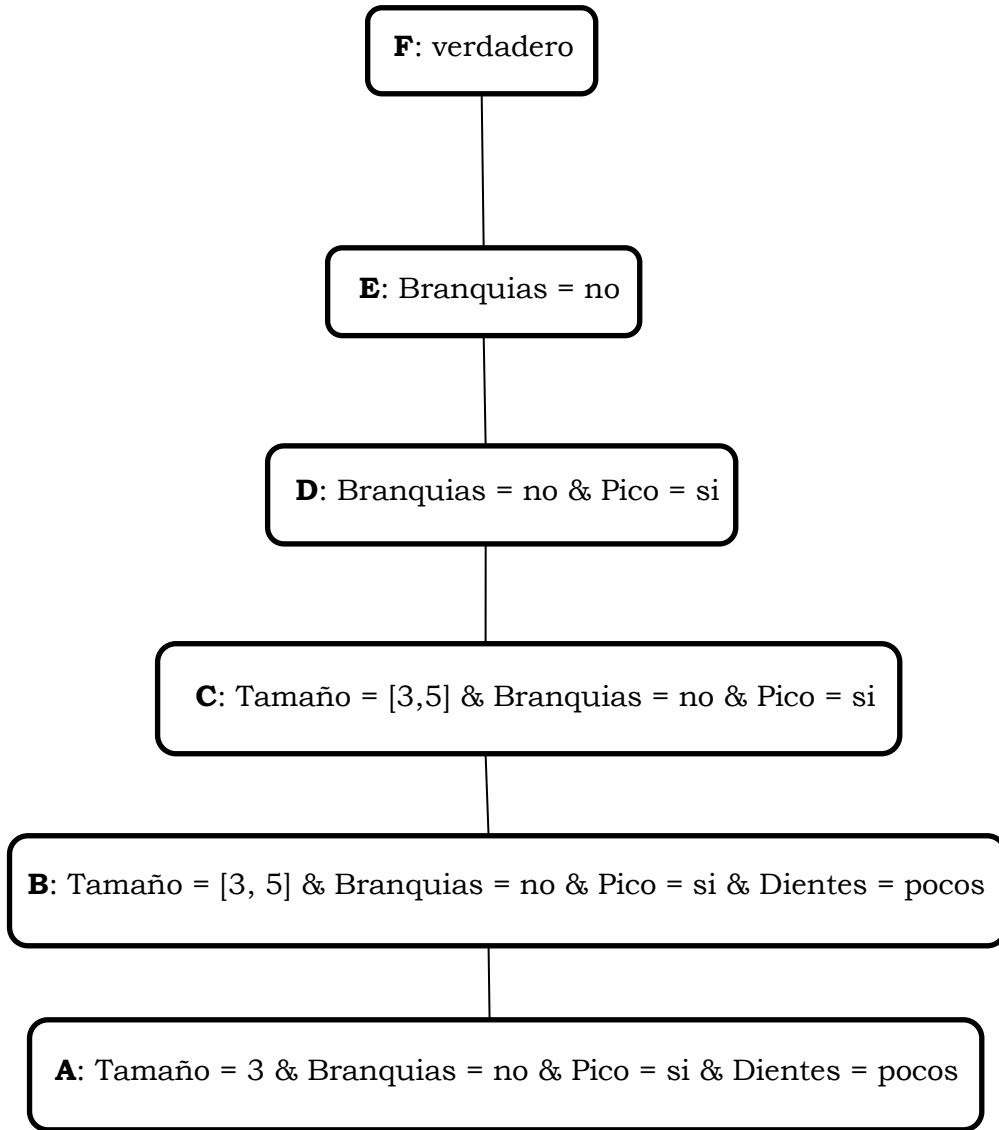
La hipótesis completa menos general es **Branquias = no  $\wedge$  Pico** como antes, pero esto cubre **n5**, y por lo tanto es inconsistente. Hay siete hipótesis consistentes generales más, ninguna de las cuales está completa:

- |  |  |
|--|--|
| Tamaño = 3                               | (cubre <b>p1</b> y <b>p3</b> )               |
| Tamaño = [3, 5] $\wedge$ Branquias = no  | (cubre todos los positivos menos <b>p2</b> ) |
| Tamaño = [3, 5] $\wedge$ Dientes = pocos | (cubre <b>p3</b> y <b>p5</b> )               |
| Branquias = no $\wedge$ Dientes = muchos | (cubre <b>p1</b> , <b>p2</b> y <b>p4</b> )   |
| Branquias = no $\wedge$ Pico = no        |  |
| Branquias = si $\wedge$ Dientes = pocos  |  |
| Pico = no $\wedge$ Dientes = pocos       |  |

Estos tres últimos ejemplos no cubren ningún ejemplo positivo.

*Tabla 5: Ejemplo de datos que no son conjuntivamente separables*

Se van a convertir las hipótesis del ejemplo de la *Tabla 5* en un camino de hipótesis de arriba abajo como se muestra en la *Figura 3*. En esta *Figura 3* se ve que en relación al ejemplo de la *Tabla 5*, el concepto **A** cubre un solo positivo (**p3**); el concepto **B** cubre un positivo más (**p5**); el concepto **C** cubre todos los positivos menos **p4**; el concepto **D** es la *GM* de todos los cinco casos positivos, pero también cubre un caso negativo (**n5**), igual que el concepto **E**.



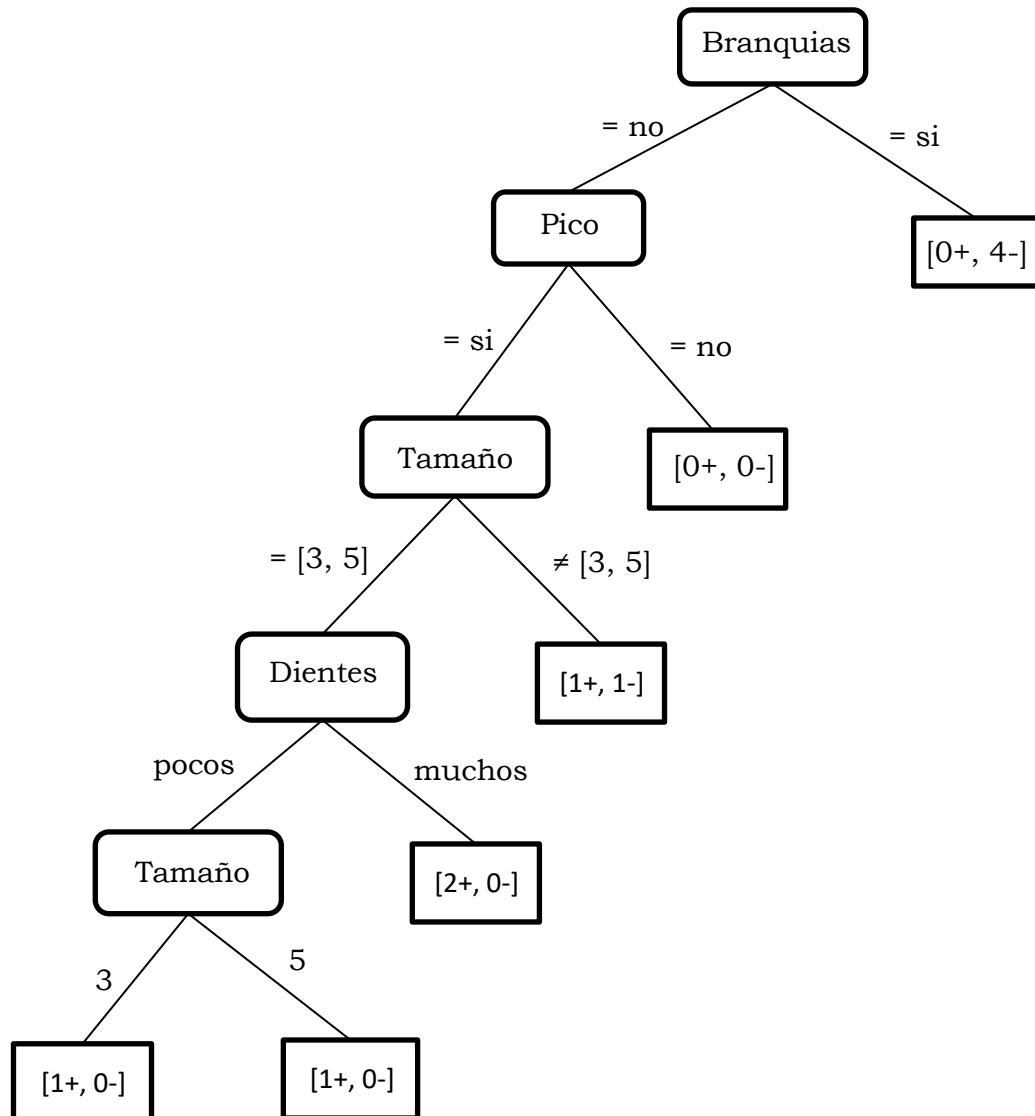
*Figura 3: Camino de hipótesis del ejemplo de la Tabla 5*

Observando este esquema de la *Figura 3*, se puede definir un concepto llamado conceptos cerrados. Esto esencialmente es la *GM* de todos los ejemplos que cubre. Por ejemplo, el concepto **D** y **E** cubren a todos los positivos y al negativo **n5**; la *GM* de esos seis ejemplos es **Branquias = no  $\wedge$  Pico = si**, el cual es la condición lógica **D**.

Dicho esto, voy a hablar ahora de los modelos basados en árboles [23]. Voy a coger el diagrama de un camino de hipótesis de la *Figura 3* y convertirlo en un

árbol con ramas y hojas como se ve en la *Figura 4*. La equivalencia de ambas figuras se ve mejor siguiendo el camino y el árbol de abajo a arriba:

1. La hoja más a la izquierda del árbol de características representa el concepto más bajo del camino, que cubre solo un caso positivo.
2. El siguiente concepto un punto más arriba en el camino generaliza el literal **Tamaño = 3** en **Tamaño = [3, 5]** por medio de una disyunción interna; la cobertura agregada (un caso positivo) se representa con la segunda hoja de la izquierda en el árbol de características.
3. Descartando la condición **Dientes = pocos** se añaden otros dos casos que se cubren.
4. Descartando la condición de “Tamaño” conjunta (o extendiendo la disyunción interna con el valor restante de “4”) se añade el último positivo, así como un negativo.
5. Descartando la condición **Pico = si** no se cubre ningún caso extra (es un caso de concepto cerrado, como se ha explicado antes).
6. Finalmente, descartando **Branquias = no** se cubren los cuatro restantes casos negativos.



*Figura 4: Diagrama del camino de hipótesis de la Figura 3 convertido en un árbol*

Los modelos basados en arboles no tienen porque solo limitarse a la clasificación, se pueden emplear para resolver casi cualquier tarea de *machine learning*, y su forma de aprendizaje común se ve reflejada en el algoritmo de la *Tabla 6*, teniendo en cuenta los conceptos de las siguientes funciones que se usarán en el algoritmo:

- *Homogeneidad(D)*: devuelve verdadero si las instancias en  $D$  son lo suficientemente homogéneas para ser etiquetadas con una sola etiqueta, y falso de cualquier otra forma.
- *Etiquetado(D)*: devuelve la etiqueta más apropiada para un conjunto de instancias  $D$
- *MejorDivision(D,F)*: devuelve el mejor conjunto de literales para ser colocados en la raíz del árbol.

Estas funciones dependen en la tarea que tenemos entre manos.

Input: datos  $D$ ; conjunto de características  $F$ .

Output: árbol de características  $T$  con hojas etiquetadas.

```

1 if Homogeneidad(D) then return Etiquetado(D);
2  $S \leftarrow \text{MejorDivision}(D, F)$ ;
3 dividir  $D$  en subconjuntos  $D_i$  de acuerdo a los literales en  $S$ ;
4 for each  $i$  do
5   if  $D_i \neq \emptyset$  then  $T_i \leftarrow \text{CrecerArbol}(D_i, F)$  else  $T_i$  es una hoja etiquetada con
Etiquetado(D);
6 end
7 return un árbol cuyas raíces están etiquetadas con  $S$  y sus hijos son  $T_i$ 
```

*Tabla 6: Algoritmo de CrecerArbol (D, F), que hace crecer un árbol de características a partir de unos datos de entrenamiento*

Este algoritmo de la *Tabla 6* es un algoritmo de divide y conquista: divide los datos en subconjuntos, construye un árbol para cada uno de ellos, y luego combina esos subárboles en un solo árbol.

#### 2.4.1.5 Árboles de decisión [24]

Los arboles de decisión no emplean la disyunción interna para características con más de dos valores, pero por el contrario permiten la ramificación en cada valor separado. También permiten el etiquetado de hojas que no sigue el orden de izquierda a derecha de las hojas. En la *Figura 5* se muestra un árbol de decisión, construido a partir del árbol de la *Figura 4*. Como se puede observar, los arboles de decisión separan perfectamente los casos positivos y los negativos.

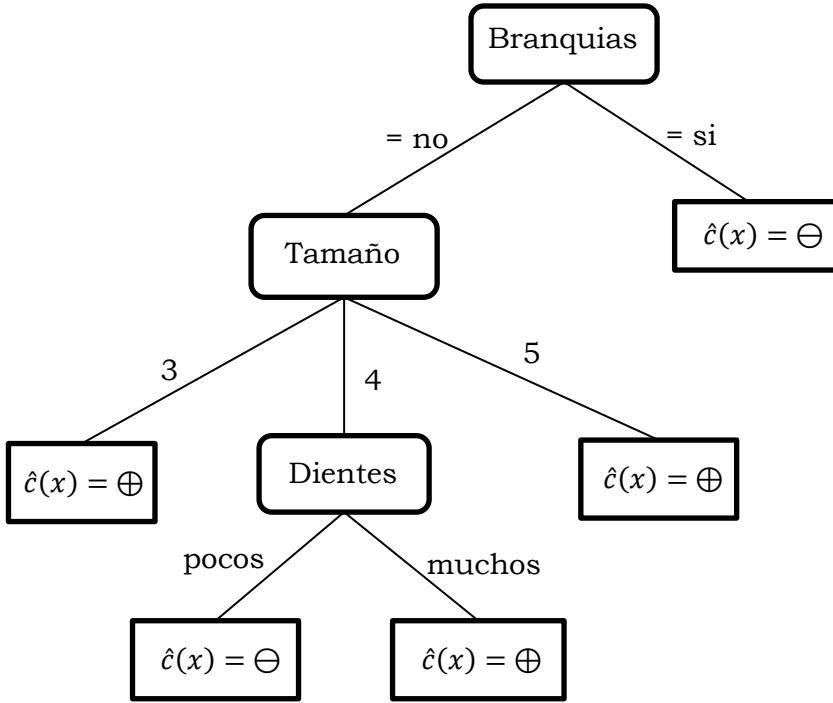


Figura 5: Un árbol de decisión basado en el árbol de la Figura 4, que aprende de los mismos datos

Como ya se dijo con anterioridad en el apartado anterior, para una simple tarea de clasificación, podemos declarar que un conjunto de instancias de datos  $D$  sea homogéneo si son todos de la misma clase, y la función  $\text{Etiquetado}(D)$  devuelve obviamente esa clase. En el paso 5 del algoritmo de la Tabla 6, se llama a la función  $\text{Etiquetado}(D)$  con un conjunto de instancias no homogéneo en caso de que algún  $D_i$  esté vacío. De esta forma se puede hacer una definición general de la función  $\text{Etiquetado}(D)$  como una función que devuelve la clase mayoritaria de las instancias en  $D^2$ . Esto lleva a decidir cómo definir la función  $\text{MejorDivisión}(D, F)$ .

Se asume por un momento que se está lidiando con características *booleanas*, de forma que  $D$  se divide en  $D_1$  y  $D_2$ . Se asume luego que tenemos dos clases, denotadas como  $D^\oplus$  las clases positivas, y  $D^\ominus$  las clases negativas, en  $D$  (e igual para  $D_1$  y demás). La cuestión entonces es como evaluar la utilidad de una característica en términos de dividir los distintos casos en clases positivas y negativas. Claramente, el mejor caso es cuando  $D_1^\oplus = D^\oplus$  y  $D_1^\ominus = \emptyset$ , o el caso en que  $D_1^\oplus = \emptyset$  y  $D_1^\ominus = D^\ominus$ . En ese caso, se dice que los dos hijos de la ramificación son puros. De esta forma, hay que medir la impureza de un conjunto de  $n^\oplus$  positivos, y  $n^\ominus$  negativos. Un principio importante a tener en cuenta es que la impureza solo debe depender de la magnitud relativa de  $n^\oplus$  y  $n^\ominus$ , y no debería cambiarse ni, aunque se multipliquen ambas clases por la misma cantidad. Dicho de otra forma, se puede definir la impureza en términos de proporción  $p = \frac{n^\oplus}{n^\oplus + n^\ominus}$ . Además, la impureza no debe tampoco cambiar si se intercambian la clase positiva y negativa o, dicho de otro modo, debe mantenerse igual si se reemplaza  $p$  con  $1 - p$ .

También se quiere una función que sea  $p = 0$  o  $p = 1$ , y que alcance su máximo cuando  $p = \frac{1}{2}$ . Las funciones que se explican a continuación se ajustan a estos términos:

- **Clase minoritaria**  $\min(p, 1 - p)$ : Esta función es referida ocasionalmente como la tasa de error, puesto que mide la proporción de casos mal clasificados si la hoja ha sido etiquetada con la clase mayoritaria; cuanto más puro el conjunto de casos, menos errores se harán. Esta medida de impureza también puede ser escrita como  $\frac{1}{2} - |p - \frac{1}{2}|$ .
- **Índice de Gini**  $2p(1 - p)$ : Esta función permite definir el error esperado si se etiqueta el caso tratado en la hoja de forma aleatoria: el caso positivo con probabilidad  $p$  y el negativo con probabilidad  $1 - p$ . La probabilidad de un falso positivo es entonces  $p(1 - p)$ , y la probabilidad de un falso negativo es  $(1 - p)p^3$ .
- **Entropía**  $-p \log_2 p - (1 - p) \log_2(1 - p)$ : Esta función permite definir la información esperada, en bits, transmitida por alguien que menciona la clase de un caso dibujado al azar; cuanto más puro el conjunto de casos, más predecible se volverá este mensaje, y menor será la información esperada.

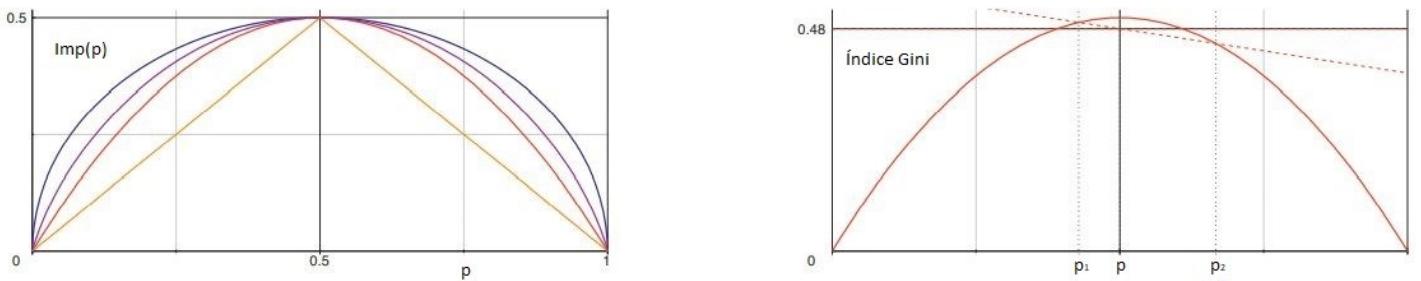


Figura 6: (Izquierda) Función de impureza trazada en contra de la probabilidad empírica de la clase positiva. (Derecha) Construcción geométrica para determinar la impureza de una bifurcación

Un trazado de estas tres medidas de impureza se puede observar en la Figura 6 (Izquierda), algunas reescaladas para que puedan llegar a su máximo en  $(0.5, 0.5)$ .

A parte de las funciones ya explicadas, he añadido también otra, que es la raíz cuadrada del índice de Gini ( $\sqrt{\text{Gini}}$ ), y que tiene una ventaja frente a las otras ya explicadas que veremos un poco más adelante.

Indicando la impureza de una sola hoja  $D_j$  como  $\text{Imp}(D_j)$ , la impureza de un conjunto de hojas mutuamente exclusivas  $\{D_1, \dots, D_l\}$  es entonces definido como un peso promedio:

$$\text{Imp}(\{D_1, \dots, D_l\}) = \sum_{j=1}^l \frac{|D_j|}{|D|} \text{Imp}(D_j)$$

Donde  $D = D_1 \cup \dots \cup D_l$ . Para una bifurcación hay una construcción geométrica que encuentra  $\text{Imp}(\{D_1, D_2\})$  dadas las probabilidades empíricas del padre y el hijo, tal como se puede observar en la Figura 6 (Derecha).

Primero se encuentran los valores de impureza  $\text{Imp}(D_1)$  e  $\text{Imp}(D_2)$  de los dos hijos en la curva de impureza (Índice de Gini).

Luego se conecta estos dos valores con una línea recta, ya que cualquier promedio ponderado de ambos valores deben estar en esa línea.

Puesto que la probabilidad empírica del padre es también un promedio ponderado de las probabilidades empíricas del hijo, con los mismos pesos,  $p$  da el punto correcto de interpolación.

Esta construcción previamente explicada funcionará con cualquier medida de impuridad representada en la *Figura 6 (Izquierda)*. Nótese que, si la clase de distribución en los padres está muy sesgada, la probabilidad empírica de ambos hijos puede acabar a la izquierda o la derecha de  $p = 0.5$  en el eje  $Y$ . Esto no es un problema, salvo por la medida de la clase minoritaria, ya que la construcción geométrica deja claro que tales bifurcaciones serán evaluadas como teniendo el mismo promedio ponderado de impuridad. Por esta razón, no se aconseja usarla como medida de impuridad.

Tomando de nuevo el ejemplo de la *Tabla 5*, se quiere encontrar la mejor característica para poner en la raíz del árbol de decisión. Las cuatro características disponibles acaban en las siguientes ramificaciones:

$$\text{Tamaño} = [3, 4, 5] \quad [2+, 0-] [1+, 3-] [2+, 2-]$$

$$\text{Branquias} = [\text{si, no}] \quad [0+, 4-] [5+, 1-]$$

$$\text{Pico} = [\text{si, no}] \quad [5+, 3-] [0+, 2-]$$

$$\text{Dientes} = [\text{muchos, pocos}] \quad [3+, 4-] [2+, 1-]$$

Entonces se calcula la impuridad de la primera ramificación: el primero es puro, y por lo tanto tiene entropía 0; el segundo tiene entropía  $-(\frac{1}{4}) \log_2(\frac{1}{4}) - (\frac{3}{4}) \log_2(\frac{3}{4}) = 0.5 + 0.31 = 0.81$ ; por último el tercero tiene entropía 1. La entropía total es entonces el promedio ponderado de estas entropías individuales, que es  $\frac{2}{10} * 0 + \frac{4}{10} * 0.81 + \frac{4}{10} * 1$ . Se realizan cálculos similares para el resto de las tres características con las entropías dadas:

$$\text{Branquias} \quad \frac{4}{10} * 0 + \frac{6}{10} * \left( -\left(\frac{5}{6}\right) \log_2\left(\frac{5}{6}\right) - \left(\frac{1}{6}\right) \log_2\left(\frac{1}{6}\right) \right) = 0.39$$

$$\text{Pico} \quad \frac{8}{10} * \left( -\left(\frac{5}{8}\right) \log_2\left(\frac{5}{8}\right) - \left(\frac{3}{8}\right) \log_2\left(\frac{3}{8}\right) \right) + \frac{2}{10} * 0 = 0.76$$

$$\text{Dientes} \quad \frac{7}{10} * \left( -\left(\frac{3}{7}\right) \log_2\left(\frac{3}{7}\right) - \left(\frac{4}{7}\right) \log_2\left(\frac{4}{7}\right) \right) + \frac{3}{10} * \left( -\left(\frac{2}{3}\right) \log_2\left(\frac{2}{3}\right) - \left(\frac{1}{3}\right) \log_2\left(\frac{1}{3}\right) \right) = 0.97$$

Se puede ver entonces que **Branquias** es una característica excelente para la que hacer la ramificación; **Dientes** es pobre; y las otras dos características están en un punto intermedio.

El cálculo para el índice *Gini* es el siguiente (todos los cálculos están realizados en el intervalo 0 a 0.5):

$$\text{Tamaño} \quad \frac{2}{10} * 2 * \left(\frac{2}{2} * \frac{0}{2}\right) + \frac{4}{10} * 2 * \left(\frac{1}{4} * \frac{3}{4}\right) + \frac{4}{10} * 2 * \left(\frac{2}{4} * \frac{2}{4}\right) = 0.35$$

$$\text{Branquias} \quad \frac{4}{10} * 0 + \frac{6}{10} * 2 * \left(\frac{5}{6} * \frac{1}{6}\right) = 0.17$$

$$\text{Pico} \quad \frac{8}{10} * 2 * \left(\frac{5}{8} * \frac{3}{8}\right) + \frac{2}{10} * 0 = 0.38$$

$$\text{Dientes} \quad \frac{7}{10} * 2 * \left(\frac{3}{7} * \frac{4}{7}\right) + \frac{3}{10} * 2 * \left(\frac{2}{3} * \frac{1}{3}\right) = 0.48$$

Como era de esperar, las dos medidas de impuridad están muy cercanas en resultados positivos.

Tabla 7: Ejemplo de cómo calcular la impuridad de unas características

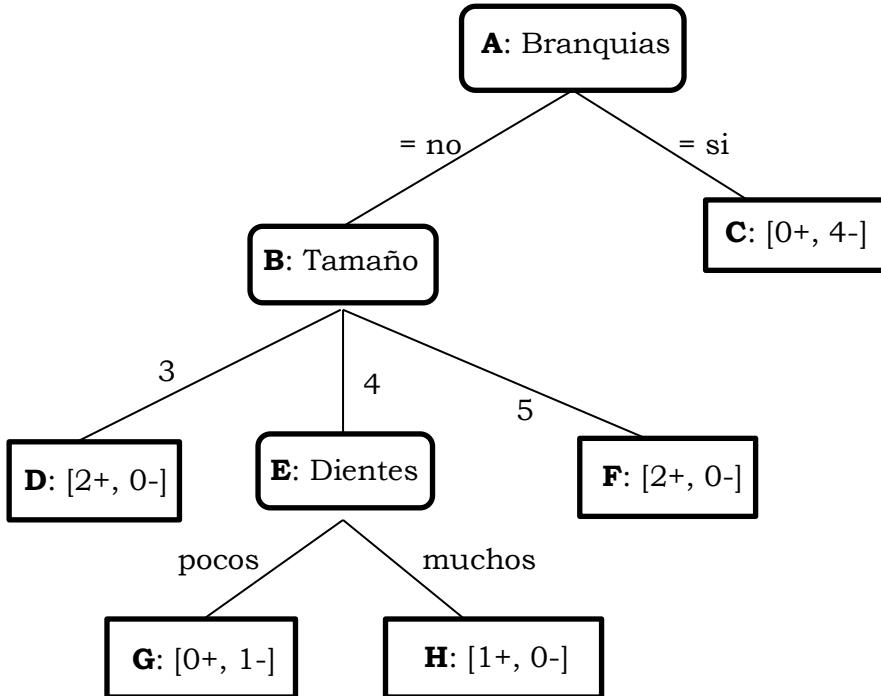


Figura 7: Árbol de decisión que ha aprendido de los datos del ejemplo de la Tabla 5

Adaptar estas medidas de impuridad a clases  $k > 2$  se realiza asumiendo las impuridades por cada clase en un modo de uno frente al resto. En particular, la entropía de  $k$ -clases se define como  $\sum_{i=1}^k -p_i \log_2 p_i$  y las  $k$ -clases del índice de *Gini* como  $\sum_{i=1}^k p_i (1 - p_i)$ . Para conseguir la calidad de una característica para dividir un nodo padre  $D$  en las hojas  $D_1, \dots, D_l$  es costumbre mirar la ganancia de pureza  $Imp(D) - Imp(\{D_1, \dots, D_l\})$ . Si la pureza se mide por entropía, se le llama el criterio de división de la ganancia de información, puesto que mide el incremento en la información sobre la clase de ganancia al incluir la característica. Sin embargo, el algoritmo de la *Tabla 6* solo compara divisiones con el mismo parente, y así podemos ignorar la impuridad del parente y buscar por la característica en el promedio ponderado de impuridad más bajo de su hijo, como en el algoritmo de la *Tabla 8*.

Con esto ya se ha instanciado el algoritmo para los árboles de decisión, así que ahora queda por ver cómo aplicar esto a los datos del ejemplo de la *Tabla 7*. Ya se ha visto que la mejor característica para poner en la raíz del árbol es *Branquias*, puesto que la condición *Branquias = si* lleva a una hoja pura  $[0+, 4-]$  con etiquetado negativo, y un hijo con una predominancia de positivo  $[5+, 1-]$ . Para la siguiente división se tiene la elección entre *Tamaño* y *Dientes*, puesto que la ramificación hacia *Pico* no hace que la impuridad decrezca. *Tamaño* resulta en una división  $[2+, 0-] [1+, 1-] [2+, 0-]$  y *Dientes* en una división  $[3+, 0-] [2+, 1-]$ ; ambas entropías y el índice *Gini* consideran el primer puro por encima del segundo. Se usa luego *Dientes* para separar el nodo impuro restante. El árbol de decisión resultante es entonces el que se muestra en la *Figura 7*.

El árbol de la *Figura 7* representa una partición del espacio como instancia, y así también asigna una clase a las 14 instancias que no eran parte del conjunto de entrenamiento; por esto se puede decir que un árbol generaliza los datos de entrenamiento. La hoja **C** deja a tres valores de características sin especificar, con un total de  $3 * 2 * 2 = 12$  posibles combinaciones de valores; cuatro de estas fueron proporcionadas como casos para el entrenamiento, de forma que la hoja **C** cubre ocho instancias sin etiquetar y las clasifica como negativas. De forma similar, dos instancias sin etiquetar son clasificadas como positivas por la hoja **D**, y dos más por la hoja **F**; una es clasificada como negativa por la hoja **G**, y la que queda se clasifica como positiva por la hoja **H**. El hecho de que más instancias sin etiquetar son clasificadas como negativas (9) que como positivas (5) es sobre todo debido a la hoja **C**: porque es una hoja muy arriba en el árbol, cubre muchas instancias. Uno puede argumentar que el hecho de que cuatro de cinco negativos tienen branquias es la regularidad más fuerte encontrada en los datos.

Input: datos  $\mathbf{D}$ ; conjunto de características  $\mathbf{F}$ .

Output: características  $f$  para dividir

```

1  $I_{min} \leftarrow 1;$ 
2 for each  $f \in F$  do
3     dividir  $D$  en subconjuntos  $D_1, \dots, D_i$  de acuerdo a los valores  $v_i$  de  $f$ ;
4     if  $Imp(\{D_1, \dots, D_i\}) < I_{min}$  then
5          $I_{min} \leftarrow Imp(\{D_1, \dots, D_i\});$ 
6          $f_{mejor} \leftarrow f;$ 
7     end
8 end
9 return  $f_{mejor}$ 
```

*Tabla 8: Algoritmo de MejorDivisionEnClase(D, F), que encuentra la mejor ramificación para el árbol de decisión*

## 2.4.2 Modelos geométricos basados en distancia [25]

De diferente forma que los modelos basados en árboles, que se vio que en esencia son modelos lógicos, los modelos basados en distancia son modelos geométricos. Estos son modelos construidos directamente sobre instancias del espacio, usando conceptos geométricos como líneas, planos, y distancias. Los modelos geométricos normalmente asumen que las instancias se describen por características con valor real  $d$  y, por lo tanto,  $\chi = \mathbb{R}^d$ . Esto se puede aplicar a cualquier valor real, aunque no sean intrínsecamente geométricos (como la edad de una persona, o la temperatura de un objeto), pero se puede representar en un sistema de coordenadas cartesianas de  $d$  dimensiones. Se pueden usar conceptos como líneas y planos para imponer una estructura en este espacio, como para establecer un sistema de clasificación. Se conoce a este tipo de modelos basados en líneas y planos geométricos como modelos lineales.

Los modelos basados en distancia son modelos geométricos que se basan relativamente en medir la distancia entre dos puntos, con, sobre todo, dos conceptos esenciales (que son las bases para entender los modelos basados en distancia) como es la distancia de *Minkowski* y la distancia métrica, que se explican en la *Tabla 9* y en la *Tabla 10* respectivamente.

Si  $\mathcal{X} = \mathbb{R}^d$ , la distancia de *Minkowski* de orden  $p > 0$  se define como:

$$\mathbf{Dis}_p(x, y) = \left( \sum_{j=1}^d |x_j - y_j|^p \right)^{\frac{1}{p}} = \|x - y\|_p$$

donde  $\|\mathbf{z}\|_p = (\sum_{j=1}^d |x_j - y_j|^p)^{\frac{1}{p}} = \|x - y\|_p$  es  **$p$ -normal** (a veces denotada como  $L_p$  normal) del vector  $\mathbf{z}$ . Me referiré normalmente a  $\mathbf{Dis}_p$  como la  **$p$ -normal**.

De esta forma, existen dos subconceptos para distintos valores de  $p$  dentro de esta distancia de *Minkowski*. Para  $p = 2$ , se define la distancia Euclídea como:

$$\mathbf{Dis}_2(x, y) = \sqrt{\sum_{j=1}^d (x_j - y_j)^2} = \sqrt{(x - y)^T (x - y)}$$

Para  $p = 1$ , se define distancia *Manhattan*, o la distancia de manzana de la ciudad:

$$\mathbf{Dis}_1(x, y) = \sum_{j=1}^d |x_j - y_j|$$

Tabla 9: Definición de la distancia de Minkowski

Dado un espacio de instancia  $\mathcal{X}$ , la distancia métrica es una función  $\mathbf{Dis}: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  de tal forma que para cualquier  $x, y, z \in \mathcal{X}$ :

1. las distancias entre un punto y él mismo son cero:  $\mathbf{Dis}(x, x) = 0$ ;
2. cualquiera de las otras distancias es mayor que cero: si  $x \neq y$  entonces  $\mathbf{Dis}(x, y) > 0$ ;
3. las distancias son simétricas:  $\mathbf{Dis}(y, x) = \mathbf{Dis}(x, y)$ ;
4. los desvíos no pueden acortar las distancias:  $\mathbf{Dis}(x, z) \leq \mathbf{Dis}(x, y) + \mathbf{Dis}(y, z)$ ;

Si la segunda condición se debilita a una no igualdad no estricta, se le llama a la función  $\mathbf{Dis}$  una función pseudo-métrica.

Tabla 10: Definición de la distancia métrica

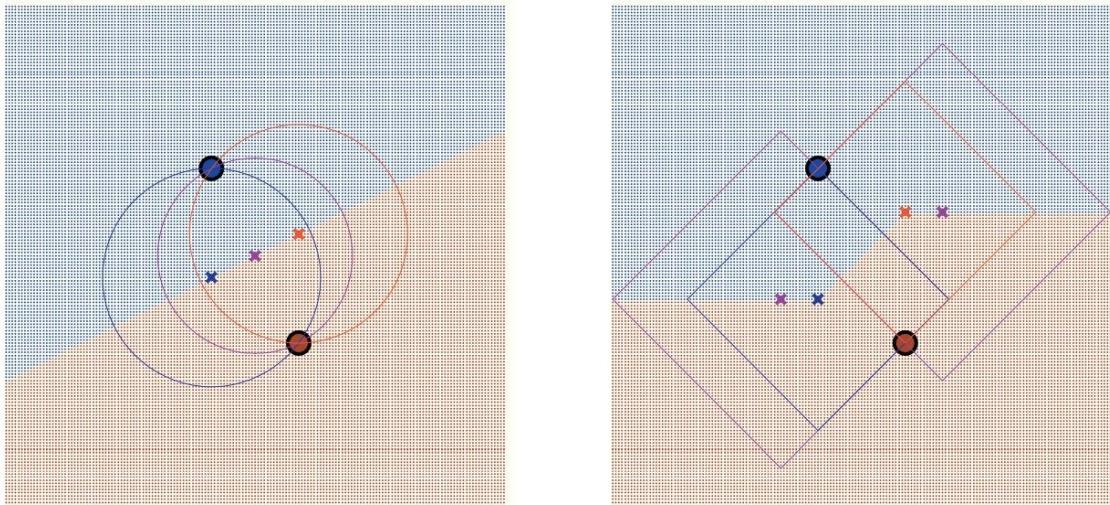
Ahora que se conocen los principios fundamentales para entender cómo se miden las distancias, se procede a considerar las ideas claves que comprenden los modelos basados en distancia. Lo más importante de estos es: formular el modelo en términos de un número de una instancia prototípica, o ejemplares; y definir las reglas de decisión en términos del ejemplar más próximo, o vecinos. Este clasificador usa el promedio de las dos clases  $\mu^{\oplus}$  y  $\mu^{\ominus}$  como ejemplares, como el resumen de todo lo que se debe saber acerca de los datos de entrenamiento para construir el clasificador. Una propiedad fundamental del promedio de un conjunto de vectores es que minimiza la suma del cuadrado de las distancias Euclídeas a esos vectores.

Minimizar la suma del cuadrado de las distancias Euclídeas de un conjunto de puntos dados es lo mismo que minimizar el promedio del cuadrado de las distancias Euclídeas. Pero, ¿Qué sucede si se deja el cuadrado aquí? ¿No sería

más natural coger el punto que minimiza la distancia Euclídea total como ejemplar? Se conoce este punto como *mediana geométrica*, ya que para los datos univariantes corresponde a la mediana o valor del medio de un conjunto de números. Sin embargo, para los datos multivariantes no hay una expresión cerrada para la mediana geométrica, que necesita ser calculada por sucesivas aproximaciones. Esta ventaja computacional es la principal razón por la que los métodos basados en distancia tienden a usar el cuadrado de la distancia Euclídea.

En ciertas situaciones tiene sentido restringir un ejemplar para que sea uno de los puntos de datos. En este caso, hablamos del *medoide*, para distinguirlo del *centroide*, que es un ejemplar que no tiene por qué ocurrir en los datos. Encontrar un *medoide* requiere que se calcule, para cada punto de datos, la distancia total a el resto de puntos, de forma que se elija el punto que lo minimiza. A pesar de la distancia métrica usada, esta es una operación de complejidad  $O(n^2)$  para  $n$  puntos, de esta forma para los *medoides* no hay una razón computacional para preferir una distancia métrica por encima de la otra.

Una vez determinados estos ejemplares, el clasificador básico lineal construye el umbral de decisión como la bisectriz perpendicular del segmento de línea conectando los dos ejemplares. Una alternativa, como un medio basado en distancia para clasificar instancias sin referencia directa al umbral de decisión es por la siguiente regla de decisión: si  $x$  es cercano a  $\mu^\oplus$  entonces se clasifica como positivo, y de cualquier otra forma como negativo; o equivalentemente, clasificar una instancia a la clase del ejemplar más cercano. Si usamos la distancia Euclídea como nuestra medida de cercanía, la geometría simple dice que se obtendrá el mismo umbral de decisión (como se observa en la *Figura 8 (Izquierda)*).



*Figura 8: (Izquierda) Umbral de decisión resultado de usar dos ejemplares con la regla de decisión de la distancia Euclídea. (Derecha) Mismo caso que la figura de la izquierda, pero usando las reglas de decisión de la distancia de Manhattan*

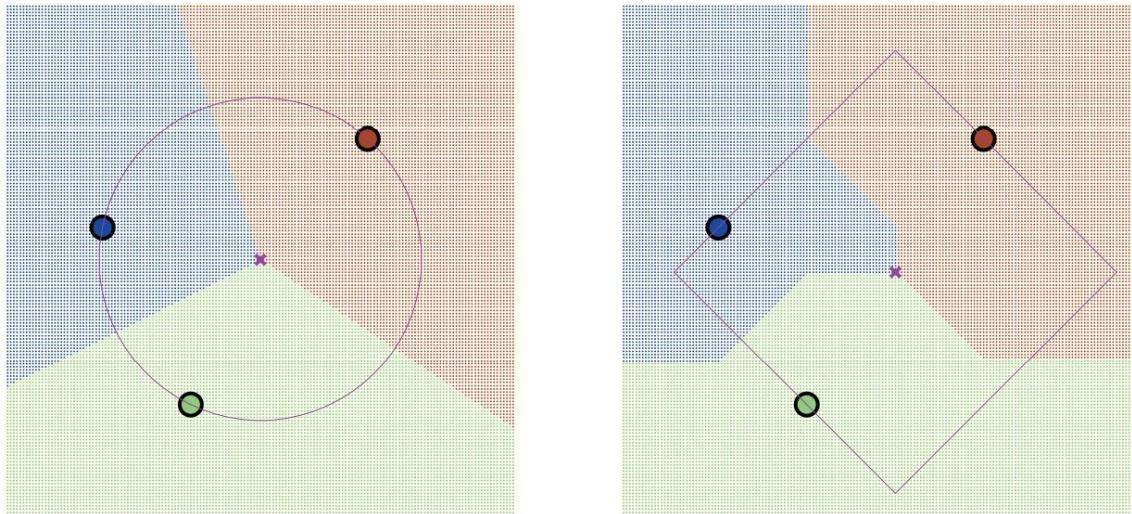


Figura 9: (Izquierda) Regiones de decisión obtenidas usando tres ejemplares y la regla de decisión de distancias Euclídeas. (Derecha) Mismo caso que el anterior, pero usando la regla de decisión de distancias de Manhattan

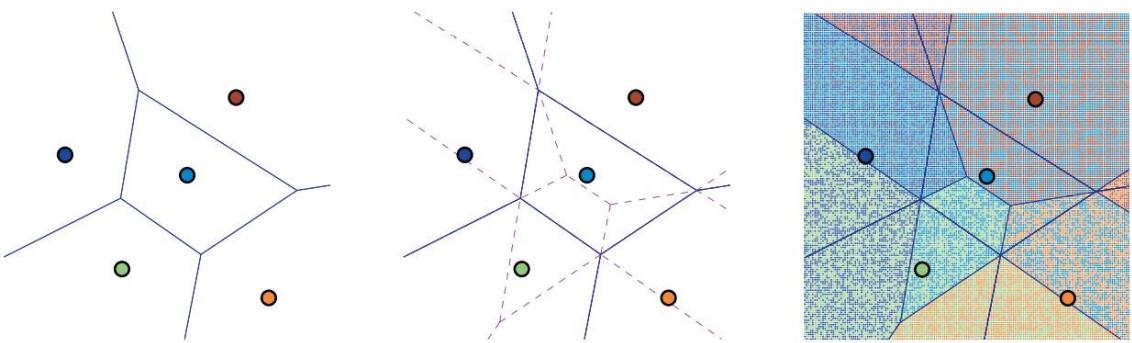


Figura 10: (Izquierda) Mosaico de Voronoi para cinco ejemplares. (Centro) Cogiendo los dos ejemplos más cercanos entre ellos se crea una mayor subdivisión en las células de Voronoi. (Derecha) Las sombras indican que ejemplos contribuyen a que células

Así pues, el clasificador básico lineal puede ser interpretado desde una perspectiva basada en distancia como construcción de ejemplos que minimicen el cuadrado de la distancia Euclídea dentro de cada clase, y luego aplicar una regla de decisión del ejemplar más cercano. Este cambio de perspectiva abre muchas nuevas posibilidades. Por ejemplo, se puede investigar cómo se vería el umbral de decisión si se usa la distancia de *Manhattan* como regla de decisión (como se observa en la Figura 8 (Derecha)). Resulta que el umbral de decisión solo puede llevarse a cabo en un número limitado de ángulos: en dos dimensiones estos son horizontales, verticales, y en (más o menos) 45 grados.

Otra consecuencia útil de cambiar a la perspectiva basada en distancias es que las reglas de decisión de los ejemplos más cercanos funcionan igualmente para más de dos ejemplos, lo que nos da una versión multiclase del clasificador lineal básico. La Figura 9 (Izquierda) ilustra estos tres ejemplos. Cada región de decisión está ahora limitada por dos segmentos de línea. Como se podría esperar, las regiones de decisión de 2 normas son más regulares que las de 1 norma: los matemáticos dicen que las de 2 normas son convexas, que significa aquí que la interpolación lineal entre cualesquiera dos puntos en la región no puede salirse nunca. Claramente, esto no se cumple para las regiones con decisiones de 1 norma (Figura 9 (Derecha)).

Incrementar el número de ejemplares más significa que algunas de las regiones se convierten en células convexas cerradas, dando lugar a lo que es conocido como mosaicos de *Voronoi*. En la *Tabla 11* se ve un ejemplo de estos mosaicos, que se van a corresponder con la *Figura 10*.

En la *Figura 10 (izquierda)* se observa un mosaico de *Voronoi* para cinco ejemplares. Cada segmento de línea es parte de una bisectriz perpendicular de dos ejemplares. Hay  $\binom{5}{2} = 10$  pares de ejemplares, pero dos de estos pares están demasiado lejos uno del otro, así que se observan solo ocho segmentos de línea en este mosaico de *Voronoi*.

Si también se tiene en cuenta el segundo ejemplar más cercano, cada célula de *Voronoi* se subdivide todavía más: por ejemplo, dado que el punto central tiene cuatro vecinos, la célula central se divide en cuatro subregiones (*Figura 10 (Centro)*). Se puede pensar en esos segmentos de línea adicionales como que forman parte del mosaico de *Voronoi* que resulta cuando el punto central es eliminado. Los otros ejemplares tienen tres vecinos inmediatos y así sus células se dividen en tres subregiones. Así se obtienen 16 regiones de decisión de ejemplares de proximidad 2, cada una definida por un par diferente de ejemplares de proximidad 1 y 2.

Finalmente, la *Figura 10 (Derecha)* muestra el sombreado de cada una de estas regiones de acuerdo a los ejemplares de proximidad 2 que abarcan.

*Tabla 11: Ejemplo de mosaicos de Voronoi*

#### 2.4.2.1 Algoritmo de k-vecinos más próximos (KNN) [26]

Ya se ha visto como generalizar los clasificadores básicos lineales para más de dos clases, aprendiendo estos un ejemplar para cada clase y usando la regla de decisión del ejemplar más próximo para clasificar los nuevos datos. De hecho, el clasificador basado en distancia más comúnmente usado es incluso más sencillo que eso: simplemente usa cada instancia del entrenamiento como un ejemplar. Consecuentemente, entrenar este clasificador requiere solamente memorizar los datos de entrenamiento. Este extremadamente simple clasificador se conoce como el clasificador del vecino más próximo. Sus regiones de decisión están compuestas por células del mosaico de *Voronoi*, con fronteras de decisiones lineales por partes seleccionadas de las fronteras de *Voronoi*.

Voy a analizar ahora las características del clasificador de vecinos más próximos. Nótese primero que, a no ser que el conjunto de entrenamiento contenga instancias idénticas para diferentes clases, se podrán separar las clases perfectamente en el conjunto de datos. Además, al elegir los ejemplares correctos se puede más o menos representar cualquier umbral de decisión, o al menos una aproximación lineal arbitrariamente cerrada por partes. De esto sale que el clasificador de vecinos más próximos tiene una parcialidad baja, pero una varianza alta: si se mueven cualquiera de los ejemplares que abarquen parte del umbral de decisión, también se cambiará la frontera. Esto sale con un riesgo de sobreajuste si los datos de entrenamiento están limitados, tienen ruido, o no son representativos.

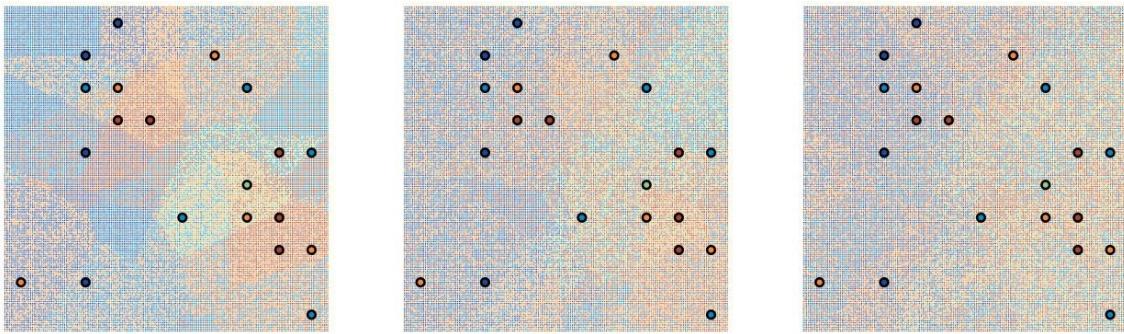
Desde el punto de vista algorítmica, entrenar clasificadores de vecinos más próximos es muy rápido, teniendo un tiempo de complejidad  $O(n)$ , puesto que la instancia necesitará compararse con todos los ejemplares para determinar cuál es la más próxima. Es posible reducir el tiempo de clasificación a expensas de incrementar el tiempo de entrenamiento al almacenar los ejemplares en una

estructura de datos más elaborada, pero esto a no tener una buena escala para un número alto de características.

De hecho, los espacios de instancias de grandes dimensiones pueden ser problemáticos por otra razón: la infame maldición de la dimensionalidad. Los espacios de grandes dimensiones tienden a ser extremadamente escasos, lo que significa que cada punto está alejado virtualmente de cualquier otro punto, y de esta forma las distancias por parejas tienden a ser poco informativas. A pesar de esto, haya sido o no afectado por la maldición de la dimensionalidad no es tan simple como el número de características, puesto que hay varias razones por las que la dimensionalidad efectiva del espacio de instancias puede ser mucho más pequeño que el número de características. De todas formas, antes de aplicar la clasificación de vecinos más próximos es buena idea trazar un histograma de distancias por parejas de una muestra para ver si son lo suficientemente variadas.

Nótese que el método de vecinos más próximos puede ser fácilmente aplicado a los problemas de regresión con una variable objetivo de valor real. De hecho, el método es completamente inadvertido para el tipo de objetivo variable y puede ser usado para los documentos de texto, imágenes, y videos, de salida. Por supuesto solo se pueden generar objetivos (o ejemplares) guardados en la base de datos de ejemplares, pero si se tiene una forma de agregar estos, se puede ir más allá de las restricciones aplicando el método de k-vecinos más próximos. En su forma más simple, el clasificador de k-vecinos más cercanos coge un voto entre los ejemplares cercanos a  $k \geq 1$  de la instancia que va a ser clasificada, y predice la clase mayoritaria. Se puede convertir esto fácilmente en una estimación probabilística devolviendo el contador de clases normalizadas como una distribución de probabilidad sobre las clases.

La *Figura 11* muestra lo explicado en un *dataset* pequeño de 20 ejemplares construido a partir de cinco diferentes clases, para  $k = 3, 5, 7$ . La distribución de clase se visualiza al asignar a cada punto de prueba, la clase de un vecino muestreado uniformemente: de esta forma en una región donde  $k = 3$  vecinos son de color rojo y uno es naranja, el sombreado es una mezcla de  $2/3$  de rojo y  $1/3$  de naranja. Mientras que para  $k = 3$  las regiones de decisión son mayormente discernibles, esto se cumple menos para  $k = 5$  y  $k = 7$ . Esto puede parecer extraño al contemplar el ejemplo de la *Tabla 11* en el que se veía el incrementar del número de regiones de decisión al incrementar el valor de  $k$ . Sin embargo, este incremento es contrarrestado por el hecho de que los vectores de probabilidad se hagan más similares entre ellos.



*Figura 11: Regiones de decisión para un clasificador de (Izquierda) 3 vecinos más próximos. (Centro) 5 vecinos más próximos. (Derecha) 7 vecinos más próximos*

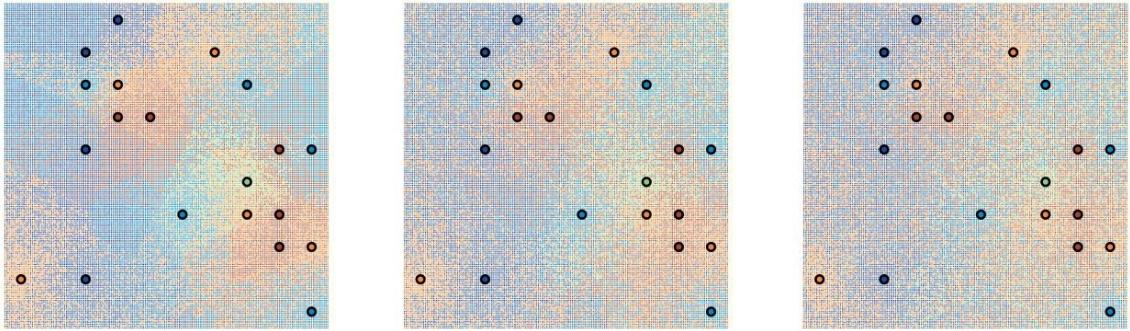


Figura 12: Basado en la Figura 11, se realiza una ponderación de las distancias sobre los datos usando el método de  $k$ -vecinos más próximos con (Izquierda) 3 vecinos más próximos. (Centro) 5 vecinos más próximos. (Derecha) 7 vecinos más próximos

Se concluye que el refinamiento del método de  $k$ -vecinos más próximos inicialmente se incrementa con el incremento de la variable  $k$ , para luego decrecer de nuevo. Además, se puede decir que la parcialidad crece y la varianza decrece con el incremento de  $k$ . No hay una receta sencilla para decidir qué valores de  $k$  son apropiados para un dado *dataset*. Sin embargo, es posible esquivar esta cuestión hasta cierto punto aplicando distancias ponderadas a los votos: esto es, cuando más cercano sea un ejemplar a la instancia que va a ser clasificada, más va a contar su voto. La Figura 12 demuestra esto, usando el recíproco de la distancia para un ejemplar como el peso de su voto. Esto difumina las fronteras de decisión, ya que el modelo aplica ahora una combinación de agrupación por medio del umbral de Voronoi, y calificando por medios de la ponderación de distancias. Para ir más lejos, dado que los pesos decrecen rápidamente para distancias largas, el efecto de incrementar  $k$  es mucho más pequeño que con la votación sin pesos. De hecho, con las distancias ponderadas podemos simplemente poner  $k = n$  y aun así obtener un modelo que haga diferentes predicciones en diferentes partes del espacio de instancias. Uno podría decir que la ponderación de distancias hace que el método de  $k$ -vecinos más cercanos se un modelo más global, mientras que sin esa ponderación (y para  $k$  más pequeñas) es más una agregación de modelos locales.

#### 2.4.3 Modelos probabilísticos [27]

En *machine learning*, aparte de los ya visto modelos lógicos y geométricos, también existen los modelos probabilísticos. Estos métodos probabilísticos pueden ser útiles para expresar las expectativas de un modelo sobre la clase de una instancia dada. Por ejemplo, un modelo lineal calibrado puede trasladar la distancia del umbral de decisión en una clase de probabilidad. Este tipo de casos son conocidos como modelos probabilísticos discriminativos. Estas modelan la distribución de probabilidad posterior  $P(Y|X)$ , donde  $Y$  es la variable objetivo, y  $X$  son las características. Dicho de otro modo, dado  $X$  se devuelve una distribución de probabilidad sobre  $Y$ .

El otro tipo de clase de modelos probabilísticos se llaman los modelos generativos. Estos modelan la distribución conjunta  $P(Y|X)$  del objetivo  $Y$  y el vector de característica  $X$ . Una vez se tiene acceso a esta distribución conjunta podemos derivar cualquier distribución condicional o marginal que envuelva a las mismas variables. En particular, dado que  $P(X) = \sum_y P(Y = y, X)$ , se puede deducir que la distribución posterior se puede obtener como:

$$P(Y|X) = \frac{P(Y|X)}{\sum_y P(Y = y, X)}$$

Alternativamente, los modelos generativos pueden ser descritos por la función de probabilidad  $P(X|Y)$ , dado que  $P(Y,X) = P(X|Y)P(Y)$  y el objetivo o distribución a priori puede ser fácilmente estimado o postulado. A estos modelos se les llama generativos porque se pueden muestrear a partir de distribuciones conjuntas para obtener nuevos puntos de datos junto con sus etiquetas. Alternativamente se puede usar  $P(Y)$  para muestrear una clase y  $P(X|Y)$  para muestrear una instancia para esa clase. En contraste, un modelo discriminativo modela  $P(Y|X)$  pero no  $P(X)$ , y por lo tanto puede ser usado para etiquetar datos, pero no para generarlos.

Dado que los modelos generativos pueden hacer cualquier cosa que haga los modelos discriminativos, puede que sean preferibles. Sin embargo, tienen también algunas fallas. Lo primero que se debe notar es que guardar las distribuciones conjuntas requiere un espacio exponencial en el número de características. Estas necesidades simplifican supuestos como la independencia entre características, que puede llevar a inexactitudes si no son válidas en un dominio particular. El criticismo más común en contra de los modelos generativos es que la exactitud al modelar  $P(X)$  puede ser conseguido a expensas de un modelado menos exacto de  $P(Y|X)$ . Sin embargo, el problema no se ha comprendido aún del todo, y hay algunas situaciones donde el conocimiento de  $P(X)$  proporciona una nueva compresión adicional del dominio.

Una de las características más atractivas de la perspectiva probabilística es que permite ver el aprendizaje como un proceso de incertidumbre reducida. Por ejemplo, una clase uniforme a priori nos dice, antes de conocer nada de las instancias que van a ser clasificadas, hay una máxima incertidumbre sobre qué clases asignar. Si la distribución posteriori después de observar la instancia es menos uniforme, hemos reducido nuestra incertidumbre en favor de una clase o la otra. Se puede repetir este proceso cada vez que recibamos nueva información, usando el posteriori obtenido del anterior paso como el priori para el siguiente paso. Este proceso puede ser aplicado, en principio, a cualquier cantidad con la que nos encontremos.

Las probabilidades no tienen por qué ser interpretadas como estimaciones de frecuencias relativas, pero pueden llevar el significado más general de grados de creencias. Consecuentemente, se puede adjuntar una distribución de probabilidad a casi cualquier cosa; no solo a características y objetivos, sino también a parámetros de modelos e incluso a los propios modelos.

Un concepto importante relacionado con los modelos probabilísticos es el Bayes-óptimo. Un clasificador es Bayes-óptimo si siempre asigna  $\arg \max_y P^*(Y|X=x)$  a una instancia  $x$ , donde  $P^*$  denota la distribución verdadera posteriori. Incluso si apenas no se sabe la verdadera distribución en una situación práctica, hay varias maneras en las que podemos hacer esto concreto. Alternativamente, la derivación de un método de aprendizaje probabilístico normalmente realiza ciertas suposiciones acerca de la verdadera distribución, lo que permite que se pruebe teóricamente que el modelo va a ser Bayes-óptimo en caso de que estas suposiciones se cumplan.

En el siguiente subapartado se va a ver los casos de características categóricas dentro de los modelos probabilísticos, lo que acabará llevando al clasificador de Bayes ingenuo.

#### **2.4.3.1 Algoritmo de Bayes ingenuo [28]**

Una pregunta binaria como puede ser “¿Es ese coche azul?” es llamado la prueba de *Bernoulli* por los estadistas. Son modeladas como una variable

binaria aleatoria cuya probabilidad de éxito está fijada para cada prueba independiente. Por encima de tal variable aleatoria, otras distribuciones de probabilidad se pueden construir. Por ejemplo, puede que se quiera adivinar cuantos de los siguientes  $n$  coches son azules: esto es gobernado por la distribución binomial. O la tarea puede ser estimar cuantos coches se ven antes de que aparezca uno alemán: este número sigue una definición geométrica.

Las variables categóricas o características (también llamadas discretas o binomiales) son ubicuas en el área de *machine learning*. Quizás la forma más común de la distribución *Bernoulli* modela la probabilidad de aparición de una palabra en un documento. Esto es, para la  $i$ -enésima palabra en el vocabulario se tiene una variable aleatoria  $X_i$  gobernada por una distribución *Bernoulli*. La distribución conjunta sobre el vector de bits  $X = (X_1, \dots, X_k)$  se llama una distribución multivariante de *Bernoulli*. Las variables con más de dos resultados también son comunes: por ejemplo, cada posición de palabra en un email corresponde a una variable categórica con  $k$  resultados, donde  $k$  es el tamaño del vocabulario. La distribución multinomial se manifiesta como un vector de conteo: un histograma del número de ocurrencias de todas las palabras del vocabulario en un documento. Esto establece un modo alternativo de modelar documentos de texto que permitan el número de ocurrencias de una palabra para influenciar la clasificación de un documento.

Ambos modelos de documentos son de uso común. A pesar de sus diferencias, ambos asumen independencia entre las ocurrencias de palabras, generalmente referidas como la suposición de *Bayes* ingenuo. En el modelo de documento multinomial, esto sigue desde el uso de una distribución multinomial, que asume que palabras en posiciones de diferentes palabras se extraen independientemente de las mismas distribuciones categóricas. En el modelo multivariante de *Bernoulli* se asumen que los bits en un vector de bits son estadísticamente independientes, lo que permite computar la probabilidad conjunta de un particular vector de bits  $(X_1, \dots, X_k)$  como el producto de probabilidades para cada componente  $P(X_i = x_i)$ . En la práctica, la independencia de suposición de estas palabras no suele ser cierta nunca. En cualquier caso, por experiencia, mientras que la suposición de *Bayes* ingenuo casi siempre lleva a estimaciones de pobre probabilidad, nunca afecta a la eficiencia de la clasificación. Esto significa que, con un umbral de clasificación elegido cuidadosamente, se suelen conseguir buenos rendimientos en la clasificación.

Se asume que se ha escogido una de las posibles distribuciones para modelar un conjunto de datos  $X$  correspondientes a correos de email, y se quieren clasificar en correos *spam* y *ham*. En un contexto de clasificación, se asume que la distribución depende de las clases, de forma que  $P(X|Y = \text{spam})$  y  $P(X|Y = \text{ham})$  tienen diferentes distribuciones. Cuanto más diferente sean estas dos distribuciones, más útiles van a ser las características  $X$  para la clasificación. De esta forma, para un email específico  $x$  se calculan ambos  $P(X = x|Y = \text{spam})$  y  $P(X = x|Y = \text{ham})$ , y aplicamos una de las posibles reglas de decisión:

$$\text{Maxima probabilidad}(MP) \rightarrow \text{predice } \arg \max_y P(X = x|Y = y);$$

$$\text{Máximo a posteriori}(MAP) \rightarrow \text{predice } \arg \max_y P(X = x|Y = y)P(Y = y);$$

$$\text{Probabilidad recalibrada} \rightarrow \text{predice } \arg \max_y w_y P(X = x|Y = y);$$

La relación entre las dos primeras reglas de decisión es que la clasificación *MP* es equivalente a la clasificación *MAP* con una clase de distribución uniforme. La tercera regla de decisión generaliza las dos primeras en que reemplaza la clase

de distribución con un conjunto de pesos aprendidos de los datos: esto hace posible que se corregir para la estimación errores en las probabilidades.

Se supone que el vocabulario utilizado contiene tres palabras  $a$ ,  $b$ , y  $c$ , y que usamos un modelo multivariante de *Bernoulli* para los emails con los parámetros

$$\theta^\oplus = (0.5, 0.67, 0.33) \quad \theta^\ominus = (0.67, 0.33, 0.33)$$

Esto significa que, por ejemplo, la presencia de  $b$  tiene el doble de posibilidades de aparecer en correos *spam* (+), comparado con *ham*.

El email que va a ser clasificado contiene las palabras  $a$  y  $b$ , pero no  $c$ , y de esta forma es descrito por el vector de bits  $x = (1, 1, 0)$ . Se obtienen las probabilidades:

$$P(X| \oplus) = 0.5 * 0.67 * (1 - 0.33) = 0.222 \quad P(X| \ominus) = 0.67 * 0.33 * (1 - 0.33) = 0.148$$

La clasificación *MP* de  $x$  es por lo tanto *spam*. En el caso de dos clases es siempre conveniente trabajar con ratios de probabilidad y posibilidades. La ratio de probabilidad puede ser calculado como  $\frac{P(x|\oplus)}{P(x|\ominus)} = \frac{0.5}{0.67} \frac{0.67}{0.33} \frac{1-0.33}{1-0.33} = \frac{3}{2} > 1$ .

Esto significa que la clasificación *MAP* de  $x$  es también *spam* si las posibilidades a priori son más de  $\frac{2}{3}$ , pero son correos *ham* si bajan de ese valor.

Por ejemplo, con un 33% de *spam* y un 67% de *ham* las posibilidades a priori son  $\frac{P(\oplus)}{P(\ominus)} = \frac{0.33}{0.67} = \frac{1}{2}$ , resultando en una posibilidad posterior de  $\frac{P(\oplus|x)}{P(\ominus|x)} = \frac{P(x|\oplus)}{P(x|\ominus)} \frac{P(\oplus)}{P(\ominus)} = \frac{3}{2} * \frac{1}{2} = \frac{3}{4} < 1$ . En este caso la ratio de probabilidad para  $x$  no es lo suficientemente fuerte para apartar la decisión lejos del priori.

Alternativamente, se puede emplear un modelo multinomial. Los parámetros de un multinomial establecen una distribución sobre las palabras en el vocabulario, como:

$$\theta^\oplus = (0.3, 0.5, 0.2) \quad \theta^\ominus = (0.6, 0.2, 0.2)$$

El email que va a ser clasificado contiene tres ocurrencias de la letra  $a$ , una sola ocurrencia de la palabra  $b$  y ninguna ocurrencia de la palabra  $c$ , por lo que se describe mediante el vector de conteo  $x = (3, 1, 0)$ . El número total de ocurrencias de palabras del vocabulario es  $n = 4$ . Se obtienen así las probabilidades:

$$P(x| \oplus) = 4! \frac{0.3^3}{3!} \frac{0.5^1}{1!} \frac{0.2^0}{0!} = 0.054 \quad P(x| \ominus) = 4! \frac{0.6^3}{3!} \frac{0.2^1}{1!} \frac{0.2^0}{0!} = 0.1728$$

La ratio de probabilidad es  $(\frac{0.3}{0.6})^3 (\frac{0.5}{0.2})^1 (\frac{0.2}{0.2})^0 = \frac{5}{16}$ . La clasificación *MP* de  $x$  es entonces de correo tipo *ham*, el contrario al del modelo multivariante de *Bernoulli*. Esto es debido principalmente a las tres ocurrencias de la palabra  $a$ , que da una alta evidencia para que sea *ham*.

Tabla 12: Ejemplo de predicciones usando el modelo de Bayes ingenuo

Nótese como la ratio de probabilidad para el modelo multivariante *Bernoulli* es producto de los factores  $\theta_i^\oplus / \theta_i^\ominus$  si  $x_i = 1$  en el vector de bits que va a ser clasificado, y  $(1 - \theta_i^\oplus) / (1 - \theta_i^\ominus)$  si  $x_i = 0$ . Para el modelo multinomial los factores son  $(\theta_i^\oplus / \theta_i^\ominus)^{x_i}$ . Una consecuencia de esto es que el modelo multinomial solo tiene en cuenta la presencia de palabras, mientras que con el modelo multivariante de *Bernoulli* la ausencia de palabras también podia marcar una

diferencia. En el ejemplo anterior de la *Tabla 12*, la ausencia de la palabra  $b$  corresponde a un factor de  $\frac{(1-0.67)}{(1-0.33)} = \frac{1}{2}$  en la ratio de probabilidad. La otra diferencia principal entre estos dos modelos es que las múltiples ocurrencias de letras son tratadas como características duplicadas en el modelo multinomial, a través del peso exponencial  $x_i$ . Esto se vuelve más claro cuando se coge el logaritmo de la ratio de probabilidad, que es  $\sum_i x_i (\ln \theta_i^\oplus - \ln \theta_i^\ominus)$ : esta expresión es lineal en  $\ln \theta_i^\oplus$  y  $\ln \theta_i^\ominus$  con  $x_i$  como pesos.

El hecho de que la ratio de probabilidad conjunta de un modelo de *Bayes* ingenuo se factorice como un producto de ratios de probabilidad de palabras individuales es una consecuencia directa de las suposiciones de *Bayes* ingenuo. En otras palabras, la tarea de aprendizaje se descompone en tareas univariantes, una para cada palabra en el vocabulario.

Una consecuencia que se infravalora bastante es que teniendo probabilidades no calibradas como aquellas que se producen por los modelos de *Bayes* ingenuo es que ambas reglas de decisión *MP* y *MAP* se vuelven inadecuadas. A menos que se tengan evidencias de que las suposiciones del modelo se satisfagan, la única cosa sensible que se puede hacer en este caso es invocar la regla de decisión de probabilidad recalibrada, que requiere que se aprenda un vector de pesos sobre las clases, de forma que se corrijan las estimaciones de errores en las probabilidades. Específicamente, se quieren encontrar los pesos  $w_i$  de forma que predecir  $\arg \max_y w_y P(X = x|Y = y)$  resulte en la menor perdida posible sobre un conjunto de pruebas. De esta forma, para dos clases, la regla de decisión de la probabilidad recalibrada puede ser reescrita como:

Un predictor positivo si  $w^\oplus P(X = x|Y = \oplus) > w^\ominus P(X = x|Y = \ominus)$  y negativo de cualquier otra forma; que es equivalente a la siguiente reinterpretación.

Un predictor positivo si  $P(X = x|Y = \oplus)/P(X = x|Y = \ominus) > w^\ominus/w^\oplus$  y negativo de cualquier otra forma.

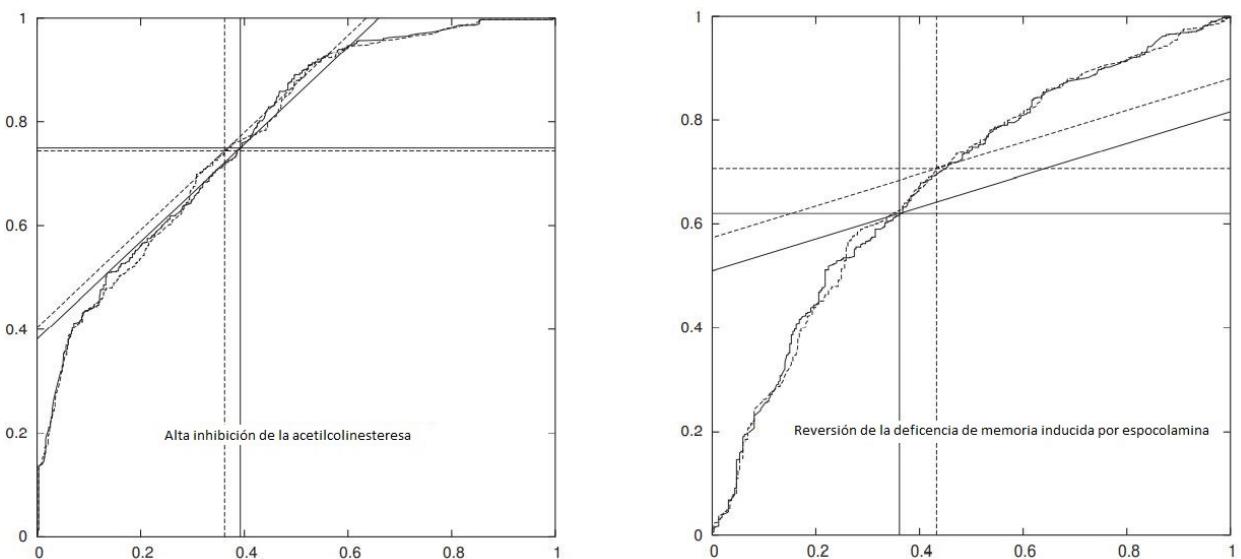


Figura 13: (Izquierda) Curvas ROC producidas por dos clasificadores de *Bayes* ingenuo. (Derecha) Mismo dominio que el anterior en un data set diferente, con un umbral del modelo multinomial *MAP* ligeramente mejor

Lo anterior dicho demuestra que en el caso de dos clases en realidad solo se tiene un grado de libertad, ya que multiplicando los pesos por una constante no afecta a la decisión. En otras palabras, en lo que se está interesado es en encontrar el mejor umbral  $t = w^\ominus/w^\oplus$  en la ratio de probabilidad, que es esencialmente el mismo problema que encontrar el mejor punto de operación en una curva *ROC* (Característica Operativa del Receptor) [29]. La solución a esto es dada por el punto en la isometría de mayor precisión. La *Figura 13* ilustra esto en dos data sets reales: en esta *Figura 13 (Izquierda)* se ve que el umbral de decisión de *MAP* es más o menos óptimo, mientras que en la *Figura 13 (Derecha)* el punto óptimo está en la esquina de arriba a la derecha. Para más de dos clases, encontrar un vector de pesos globalmente óptimo es computacionalmente intratable, lo que significa que tenemos que recurrir a un método heurístico.

Entrenar un modelo probabilístico normalmente envuelve estimar los parámetros de distribución usados en el modelo. El parámetro de una distribución de *Bernoulli* puede ser estimado contando el número de éxitos  $d$  en  $n$  pruebas y ajustando  $\hat{\theta} = d/n$ . En otras palabras, se cuentan, para cada clase, cuantos correos contienen la palabra en cuestión. La estimación de dichas frecuencias relativas es usualmente suavizada al incluir el pseudo conteo, que representa el resultado de pruebas virtuales de acuerdo a algunas distribuciones fijas. En el caso de una distribución *Bernoulli*, la operación de suavizado más común es la corrección de *Laplace*, que involucra dos pruebas virtuales, una de las cuales es el resultado en el éxito, y otra en el fracaso. Consecuentemente, la estimación de la frecuencia relativa se cambia a  $(d + 1)/(n + 2)$ . Desde una perspectiva *Bayesiana*, esto equivale a adoptar un a priori uniforme, representando la creencia inicial de que el éxito y el fracaso son similares. Si resulta apropiado, se puede reforzar la influencia del a priori incluyendo un largo número de pruebas virtuales, lo que significa se necesitan más datos para alejar la estimación del a priori. Para el suavizado de la distribución categórica se añade un pseudo conteo para cada una de las  $k$  categorías, que lleva la estimación suavizada  $(d + 1)/(n + k)$ . La m-estimación generaliza esto más allá al hacer el número total de pseudo-conteos  $m$  y la forma en la que están distribuidas sobre las categorías en parámetros. La estimación para la  $i$ -nésima categoría se define como  $(d + p_i m)/(n + m)$ , donde  $p_i$  es una distribución sobre las categorías.

E-mail	#a	#b	#c	Clase	E-mail	a?	b?	c?	Clase
$e_1$	0	3	0	+	$e_1$	0	1	0	+
$e_2$	0	3	3	+	$e_2$	0	1	1	+
$e_3$	3	0	0	+	$e_3$	1	0	0	+
$e_4$	2	3	0	+	$e_4$	1	1	0	+
$e_5$	4	3	0	-	$e_5$	1	1	0	-
$e_6$	4	0	3	-	$e_6$	1	0	1	-
$e_7$	3	0	0	-	$e_7$	1	0	0	-
$e_8$	0	0	0	-	$e_8$	0	0	0	-

Tabla 13: (Izquierda) Dataset pequeño de emails descritos por un vector de conteo. (Derecha) El mismo data set descrito por un vector de bits

Se muestran ahora como se han podido obtener los parámetros vectores del ejemplo de la *Tabla 13*. Se consideran los siguientes emails con las cinco palabras  $a$ ,  $b$ ,  $c$ ,  $d$ , y  $e$ :

$e_1$ : b d e b b d e	$e_5$ : a b a b a b a e d
$e_2$ : b c e b b d d e c c	$e_6$ : a c a c a c a e d
$e_3$ : a d a d e a e e	$e_7$ : e a e d a e a
$e_4$ : b a d b e d a b	$e_8$ : d e d e d

Se dice que los emails a la izquierda son *spam* y los de la derecha son *ham*, y de esta forma se usan como un conjunto pequeño de entrenamiento para entrenar nuestro clasificador de *Bayes*. Primero, se decide que  $d$  y  $e$  son palabras de parada, que son muy útiles para transmitir información de clase. Las palabras restantes  $a$ ,  $b$ , y  $c$  constituyen el vocabulario.

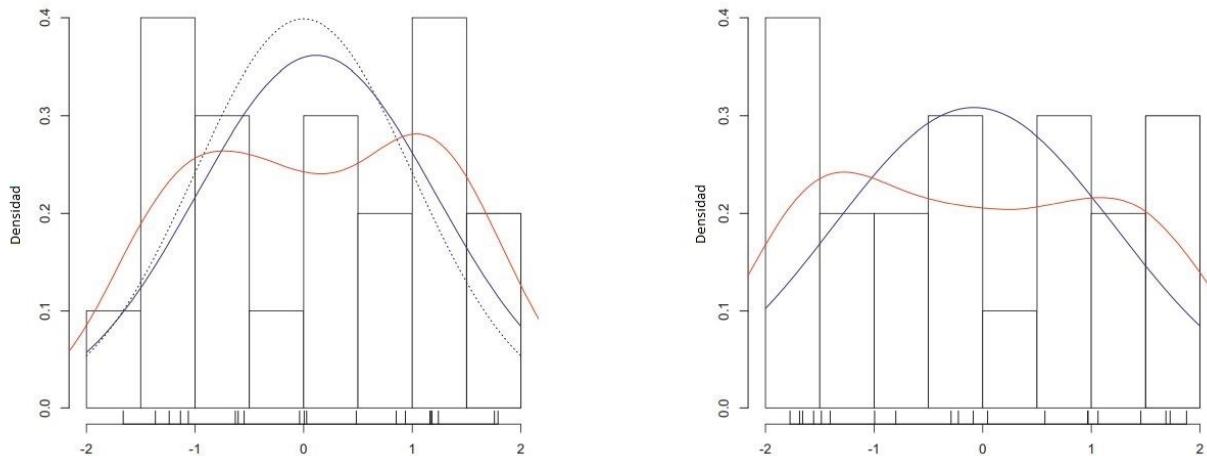
Para el modelo multinomial, se representa cada email como un vector de conteo, tal como se muestra en la *Tabla 13 (Izquierda)*. Para estimar los parámetros del multinomial, sumamos los vectores de conteo para cada clase, que da como resultado el vector  $(5, 9, 3)$  para *spam* y  $(11, 3, 3)$  para *ham*. Para suavizar estas estimaciones de probabilidades se añade un pseudo conteo para cada palabra del vocabulario, lo que trae el número total de ocurrencias de las palabras del vocabulario a 20 para cada clase. El vector de parámetros estimados es entonces  $\hat{\theta}^\oplus = \left(\frac{6}{20}, \frac{10}{20}, \frac{4}{20}\right) = (0.3, 0.5, 0.2)$  para *spam* y  $\hat{\theta}^\ominus = \left(\frac{12}{20}, \frac{4}{20}, \frac{4}{20}\right) = (0.6, 0.2, 0.2)$  para *ham*.

En el modelo multivariante *Bernoulli*, los emails son representados por vectores de bits, como se muestra en la *Tabla 13 (Derecha)*. Añadir el vector de bits para cada clase resulta en  $(2, 3, 1)$  para *spam* y  $(3, 1, 1)$  para *ham*. Cada conteo se divide por el número de documentos en una clase, de forma que conseguir una estimación de la probabilidad de un documento que contiene una palabra particular del vocabulario. El suavizado de la probabilidad ahora significa añadir dos pseudo documentos, uno conteniendo cada palabra y uno conteniendo ninguno de ellos. Esto resulta en los parámetros de vectores estimados  $\hat{\theta}^\oplus = \left(\frac{3}{6}, \frac{4}{6}, \frac{2}{6}\right) = (0.5, 0.67, 0.33)$  para *spam* y  $\hat{\theta}^\ominus = \left(\frac{4}{6}, \frac{2}{6}, \frac{2}{6}\right) = (0.67, 0.33, 0.33)$  para *ham*.

*Tabla 14: Ejemplo para entrenar un modelo de Bayes ingenuo*

Existen muchas otras variaciones del clasificador de *Bayes* ingenuo. De hecho, lo que es normalmente comprendido como el clasificador de *Bayes* ingenuo no emplea ni un modelo *Bernoulli* multinomial ni uno multivariante, pero si emplea un modelo categórico multivariante. Esto significa que las características son categóricas, y la probabilidad de la  $i$ -enésima característica cogido en su  $i$ -enésimo valor para el caso de la clase  $c$  se da por  $\theta_{il}^{(c)}$ , bajo la restricción de que  $\sum_{l=1}^{k_i} \theta_{il}^{(c)} = 1$ , donde  $k_i$  es el número de valores de la característica  $i$ -enésima. Estos parámetros pueden ser estimados al suavizar las frecuencias relativas en el conjunto de entrenamiento, como en el caso de *Bernoulli* multivariante. Se vuelve a tener de nuevo que probabilidad conjunta del vector de características es el producto de la probabilidad de características individuales, y por lo tanto  $P(F_i, F_j | C) = P(F_i | C)P(F_j | C)$  para todos los pares de características y para todas las clases.

Otra extensión del modelo de *Bayes* ingenuo es requerida cuando alguna de las características es de valor real. Una opción es discretizar las características de valor real en un estado de pre proceso; y otra opción es asumir que el valor de las características está normalmente distribuido dentro de cada clase. En este contexto, vale la pena notar que la suposición de *Bayes* ingenuo se reduce a asumir una matriz de covarianza diagonal dentro de cada clase, de forma que cada característica puede ser tratada de forma independiente. Una tercera opción que también se usa en la práctica es la probabilidad de clase condicional de cada característica por un estimador de densidad no paramétrico. Esta opción se ilustra en la *Figura 14*.



*Figura 14:* (Izquierda) Ejemplo de tres estimadores de densidad en 20 puntos muestreados a partir de una distribución normal con media 0 y varianza unitaria. (Derecha) Aquí, los 20 puntos son muestreados uniformemente a partir de [-2, 2], y los métodos no paramétricos son más eficientes

En resumen, el modelo de *Bayes* ingenuo es un modelo popular para lidiar con datos textuales, categóricos, y mezcla de categóricos/valores reales. Su principal deficiencia como modelo probabilístico (estimaciones de probabilidad mal calibradas) es superada por un rendimiento que resulta en una buena clasificación. Otra aparente paradoja de *Bayes* ingenuo es que no es *Bayesiana* del todo. Por un lado, se ha visto que la pobre probabilidad estimada necesita el uso de probabilidades reponderadas, lo que evita que se usen las reglas de *Bayes*. En segundo lugar, al entrenar un modelo de *Bayes* ingenuo se usan los parámetros de estimación de probabilidad máxima, mientras que en un enfoque *bayesiano* completo no se comprometería con un valor de parámetro particular, sino que emplean una distribución completa a posteriori.

#### 2.4.4 Modelos ensamblados [30]

Los modelos de ensamblado son aquellos que unen varios modelos de *machine learning* en uno (en este caso, varios modelos de árboles de decisión), creando un nuevo modelo o técnica más potente, dejando atrás a muchos de los otros métodos que se pueden usar dentro del *machine learning*. Sin embargo, no hay bien que por mal no venga, y estas técnicas de unión de modelos tienen una alta complejidad algorítmica y de modelo.

Estas técnicas de modelos ensamblados tienen una historia rica y diversa. La motivación principal de estas técnicas viene de la teoría de aprendizaje computacional, por un lado, y las bases de las matemáticas estadísticas por otro lado. Es bien conocido estadísticamente que las mediciones promedias pueden dar lugar a una estimación más estable y fiable, ya que reducimos la

influencia de fluctuaciones aleatorias en mediciones individuales. De esta forma, si se quiere construir un ensamblado de modelos ligeramente diferentes de los mismos datos de entrenamiento, es posible reducir de la misma forma la influencia de fluctuaciones aleatorias en modelos individuales. La pregunta clave entonces es como conseguir diversidad entre estos modelos distintos. Esto se puede conseguir entrenando modelos con subconjuntos aleatorios de datos, o incluso construyéndolos desde subconjuntos aleatorios de características ya disponibles.

En esencia, los modelos ensamblados tienen dos cosas en común:

- Construyen múltiples y diversos modelos de versiones adaptadas de datos de entrenamiento.
- Combinan la predicción de estos modelos de alguna forma, siempre calculando la media o votando la mejor elección realizada por los modelos individuales.

Sin embargo, se debe enfatizar que estos puntos en común abarcan un espacio largo y diverso, y que de forma correspondiente debemos esperar que algunos métodos sean prácticamente muy distintos, aunque superficialmente sean similares. Por ejemplo, se genera una gran diferencia si los datos de entrenamiento son adaptados para que la próxima iteración tenga en cuenta las predicciones de los modelos anteriores o no.

#### 2.4.4.1 Bosques aleatorios [31]

Para comprender el método de bosques aleatorios, se tiene que entender que es el llamado método *bagging*, también conocido como agregación de *bootstrap* [32]. Este es un método de ensamblado simple pero muy eficiente que crea diversos modelos a partir de muestras diferentes aleatorias de los *datasets* originales. Estas muestras son recogidas uniformemente con reemplazamiento, y son conocidas como muestras *bootstrap*. Debido a que estas muestras se recogen con reemplazos, contendrán casi siempre muestras duplicadas, y de esta forma algunos de los puntos de los datos originales se perderán incluso si estas muestras *bootstarp* tienen el mismo tamaño de los data sets originales. Esto es justo lo que se quiere lograr, puesto que las diferencias entre las muestras *bootstarp* van a crear diversidad entre los modelos en el ensamblado.

Para capturar la idea de cómo de diferentes pueden ser las muestras *bootstrap*, podemos calcular la probabilidad de que un punto particular de los datos no esté seleccionado para una muestra *bootstrap* de tamaño  $n$  como  $\left(1 - \frac{1}{n}\right)^n$ , siendo para  $n = 5$ ,  $\frac{1}{3}$ , y tiene límite de  $\frac{1}{e} = 0.368$  para  $n \rightarrow \infty$ . Esto significa que



Figura 15: (Izquierda) Un ensamblado de cinco clasificadores lineales básicos construidos con muestras *bootstrap* con el método *bagging*, siendo la regla de decisión la mayoría de voto. (Derecha) Si se transforman los votos en probabilidades, se ve como el ensamblado es un modelo grupal: cada instancia del segmento del espacio obtiene una probabilidad ligeramente diferente

cada muestra *Bootstrap* tiene altas probabilidades de dejar fuera alrededor de 1/3 de los puntos de los datos.

Input: data set  $\mathbf{D}$ ; tamaño del ensamblado  $T$ ; algoritmo de aprendizaje  $\mathcal{A}$ .

Output: ensamblado de modelos cuya predicción se combinará a través de votación o de hacer un promedio.

```

1 for  $t = 1$  a  $T$  do
2   construir una muestra bootstrap  $\mathbf{D}_t$  a partir de  $\mathbf{D}$  muestreando  $|\mathbf{D}|$  puntos
    de   datos con reemplazamiento;
3   ejecutar  $\mathcal{A}$  en  $\mathbf{D}_t$  para producir un modelo  $\mathbf{M}_t$ ;
4 end
5 return  $\{\mathbf{M}_t | 1 \leq t \leq T\}$ 
```

*Tabla 15: Algoritmo Bagging( $D, T, A$ ), que entrena un ensamblado de modelos a partir de muestras bootstrap*

El algoritmo de la *Tabla 15* proporciona las bases del algoritmo de *bagging*, que devuelve el ensamblado como conjunto de modelos. Como se ve, se pueden elegir combinar las predicciones a partir de diferentes modelos por medio de votación (que se basa en que la clase que ha sido predicha por la mayoría de los modelos gana), o a través de un promedio, que es más apropiado si el **output** de las bases clasificadoras son marcas o probabilidades. Un ejemplo de esto se da en la *Figura 15*, en la que se han entrenado cinco clasificadores básicos en muestras *bootstrap* a partir de 20 casos positivos y 20 negativos. Se pueden ver la diversidad de los cinco clasificadores lineales, ayudado por el hecho de que los *datasets* son bastante pequeños.

La *Figura 15* muestra la diferencia entre combinar predicciones a través de votación (como en la *Figura 15 (Izquierda)*), y crear un clasificador probabilístico haciendo un promedio (como en la *Figura 15 (Derecha)*).

Este proceso de *bagging* es particularmente útil cuando se combina con los modelos basados en árboles, los cuales son bastante sensibles a las variaciones en los datos de entrenamiento. Cuando se aplica el método a los modelos basados en árboles, el *bagging* se suele combinar con otra idea: construir un árbol a partir de un subconjunto de características aleatorio diferentes, un proceso al que se le llamará *muestreo subespacial*. Esto promueve aún más la diversidad en el ensamblado, y tiene la ventaja adicional de que el tiempo de entrenamiento de cada árbol se reduce. El método de ensamblado resultante es el llamado *bosques aleatorios*, y el algoritmo de este se ve en la *Tabla 16*.

Input: data set  $\mathbf{D}$ ; tamaño del ensamblado  $T$ ; subespacio de dimensión  $d$ .

Output: ensamblado de modelos basados en árboles cuya predicción se combina a través de la votación o de hacer el promedio.

```

1 for  $t = 1$  a  $T$  do
2   construir una muestra bootstrap  $\mathbf{D}_t$  a partir de  $\mathbf{D}$  muestreando  $|\mathbf{D}|$  puntos
    de   datos con reemplazamiento;
3   seleccionar  $d$  características al azar y reducir la dimensionalidad de  $\mathbf{D}_t$ 
    de forma acorde;
4   entrenar un modelo basado en árboles  $\mathbf{M}_t$  en  $\mathbf{D}_t$  sin podar
```

```
5 end  
6 return  $\{M_t | 1 \leq t \leq T\}$ 
```

*Tabla 16: Algoritmo de BosqueAleatorio( $D, T, d$ ), que entrena un ensamblado de modelos basados en árboles a partir de una muestras bootstrap y subespacios aleatorios*

Puesto que un árbol de decisión es un modelo de agrupación cuyas hojas crean una partición del espacio en instancia, igual hacen los bosques aleatorios: su partición en el espacio de instancias correspondientes es en esencia la intersección de la partición de árboles individuales en el ensamblado. Mientras que los bosques aleatorios son de esta manera más finos que la mayoría de particiones de árboles, puede ser en principio mapeado de nuevo a un modelo basado en árboles individual (puesto que la intersección corresponde a la combinación de ramas de dos árboles diferentes).

## 3 Desarrollo

Este capítulo está dedicado a describir el desarrollo del Trabajo realizado. Para ello, se va a explicar la arquitectura general del proyecto, así como las herramientas usadas y el código implementado.

### 3.1 Arquitectura del proyecto

El proyecto va a tener una estructura un poco más compleja de lo que se había estimado al principio. Va a consistir en dos modelos separados en dos clases distintas. En la primera clase, se pretende hacer una clasificación y separación entre emails, con los *datasets* correspondientes de emails que incluyen mensaje que son de tipo malicioso y no; además, antes de clasificar el email en si es o no malicioso, pretendo comprobar si dicho correo de texto tiene una *url*, lo que nos lleva a la segunda clase. Esta segunda clase se activaría cuando se detecta una *url* en dicho email, para analizar si dicha *url* lleva o no a una página web maliciosa, lo que indicaría que si el texto en sí, junto a la *url*, se consideran maliciosos, el email es de tipo *phishing*. El proyecto se separará en aquellos datos que sean de entrenamiento para el algoritmo, y aquellos que sean de prueba. Los *datasets* serán pre procesado, y de ellos se extraerán un conjunto de características que nos servirán para poder detectar mejor si tiene o no contenido que pueda ser *phishing* o maliciosos. Un vistazo general del proyecto a través de un esquema se representa en la *Figura 16*.

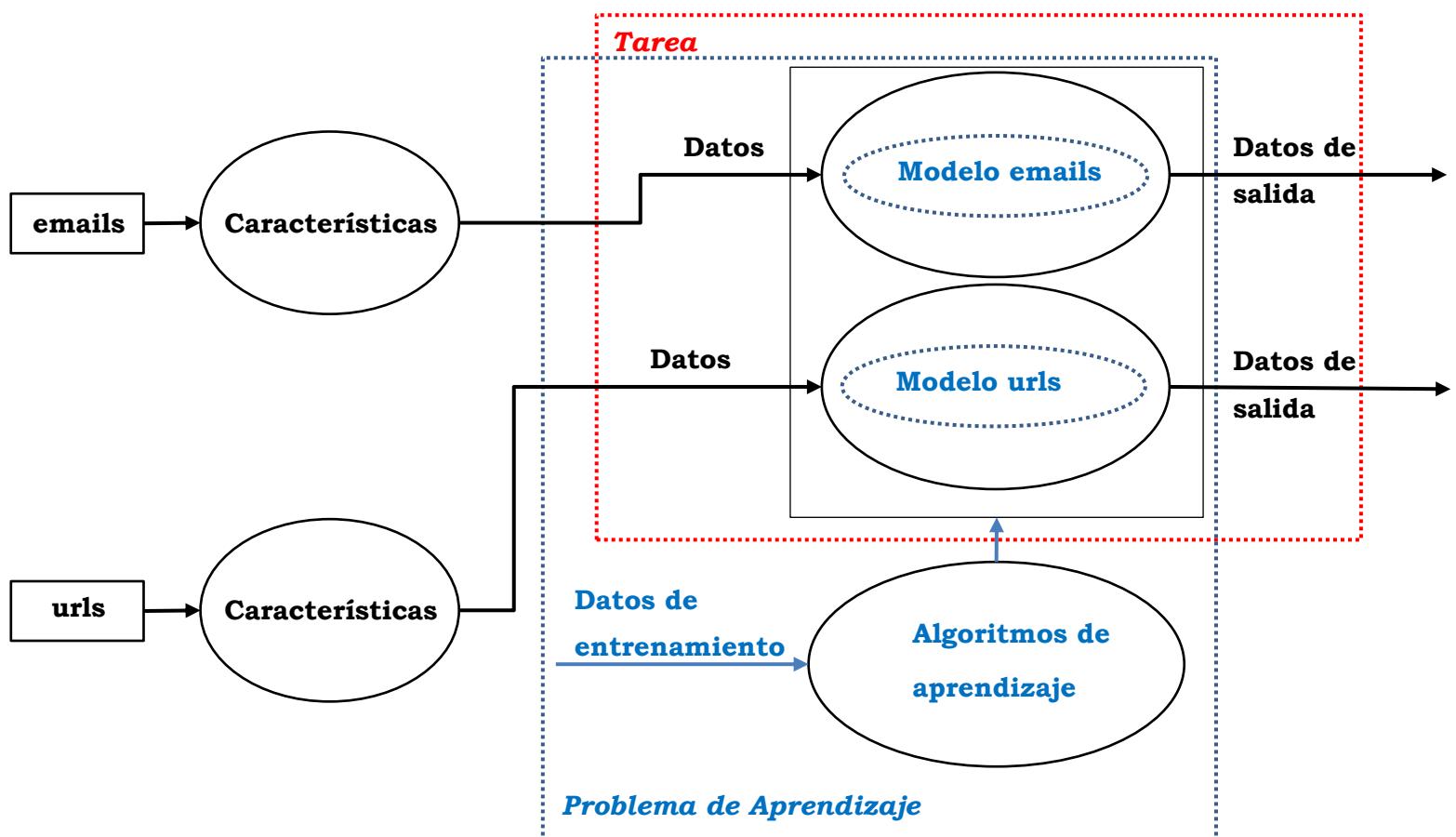


Figura 16: Arquitectura general del proyecto

Como observación secundaria, en este esquema de la *Figura 16*, se ha ahorrado diferenciar los algoritmos de aprendizaje de cada modelo, ya que para ambos modelos voy a usar los mismos algoritmos. Además, se diferenciaría entre los datos de entrenamiento de las *urls* y los de los emails. El simple hecho de ahorrar estos datos es por no hacer un esquema excesivamente complejo y lioso.

Una vez creados y entrenados los modelos, el objetivo sería poder aplicar estos modelos a casos reales. Así, lo que haría en esta etapa sería acceder a una cuenta personal de *Gmail* que he creado específicamente para este proyecto, y de ahí leer y extraer a un archivo *csv* los correos de la bandeja de entrada de esta cuenta de *Gmail*, almacenados cada uno en distintas filas y que me enviaré desde mi cuenta normal. Este documento *csv* luego será enviado a los modelos de *Machine Learning* creados para que sean clasificados. El mensaje de texto primero pasará por un método auxiliar para detectar si el correo contiene o no una o más *urls* adjuntas. Si no las tienen, se procederá a un análisis del texto de dichos correos con el modelo creado para emails, para analizar si tienen o no contenido típico de correos *spam*, y poder clasificarlo en emails *spam*, o *ham*. Si los emails contienen una o más *urls*, se analizarán con el modelo específico creado para *urls*, y si llevan a sitios maliciosos, se clasificarán automáticamente como emails de tipo *phishing*; si no, se procederá al análisis del contenido del mensaje del email, y se clasificará en emails de tipo *ham* o *spam*. La estructura de esta parte sería tal como se muestra en la *Figura 17*.

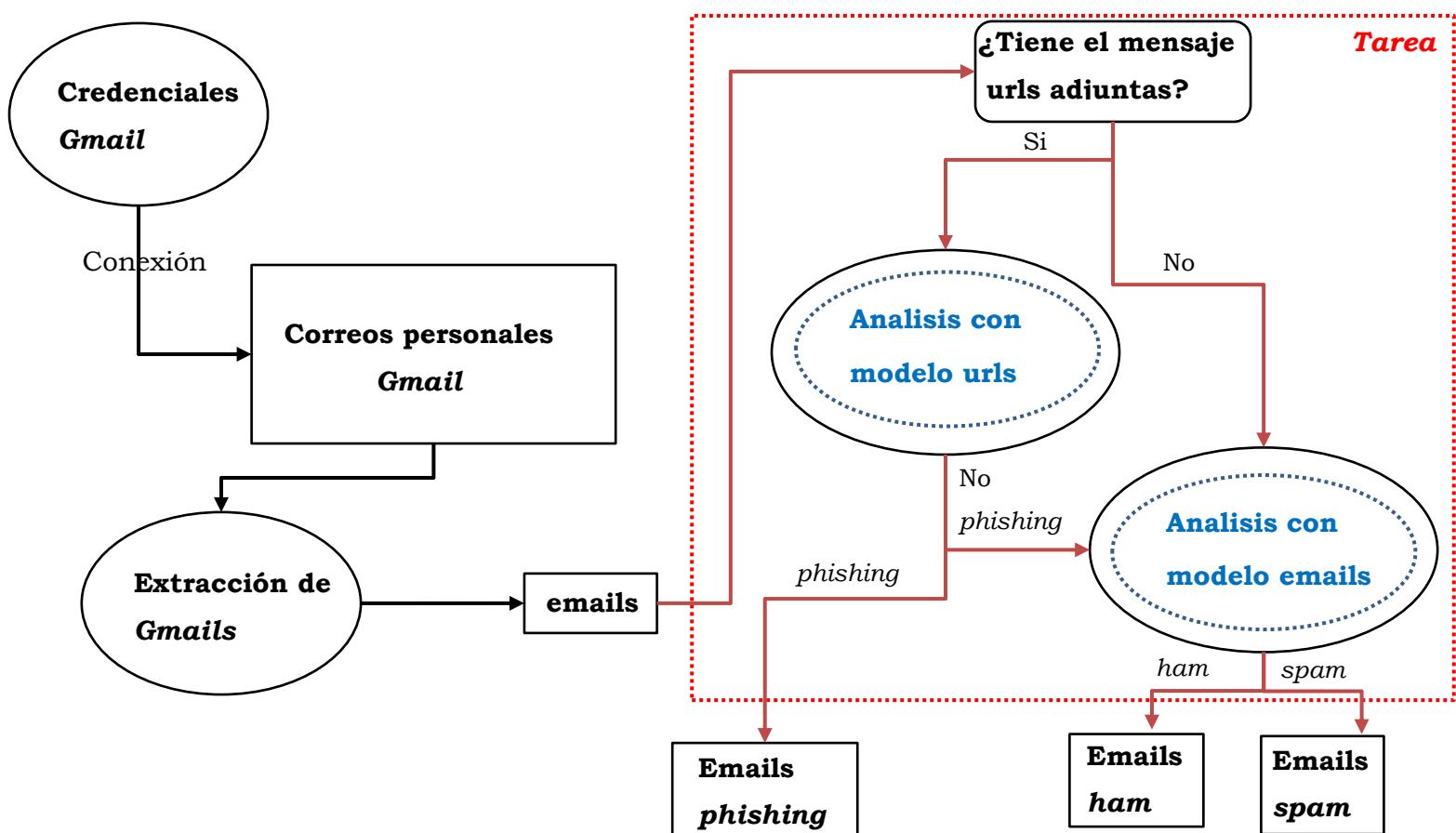
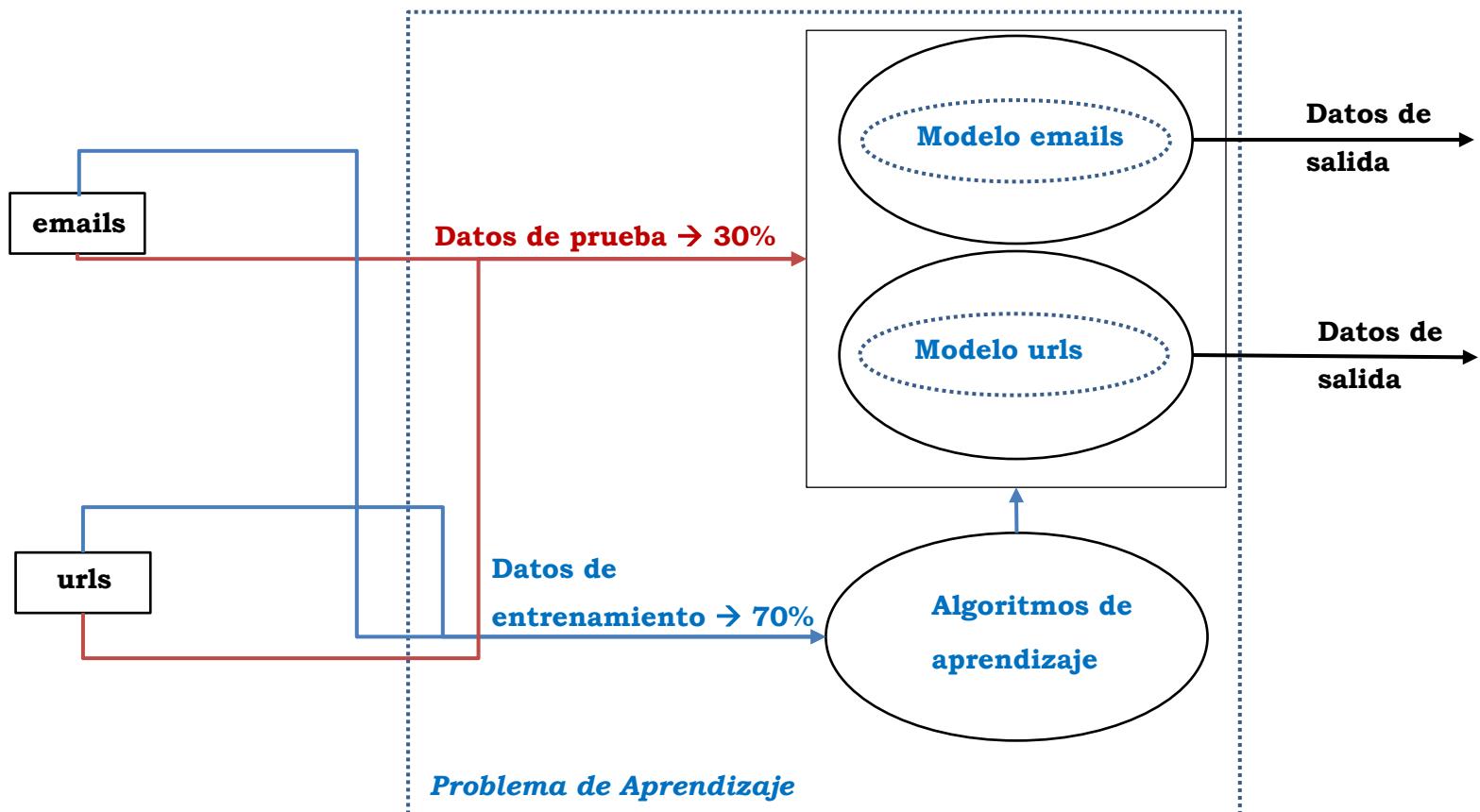


Figura 17: Arquitectura para la lectura y clasificación de los correos de Gmail

### 3.1.1 Datasets

En cuanto *datasets* que he usado, tras buscar mucho en Internet, no he podido conseguir tal cual emails con contenido phishing; pero al final, este tipo de emails son emails *spam* con *urls* que llevan a páginas externas con contenido dañino para el usuario que los abre. Por lo tanto, en mi razonamiento he pensado en entrenar a dos modelos con los mismos algoritmos, cada uno con los *datasets* correspondientes: unos de los modelos con tres *datasets* correspondientes a emails seguros, y a emails *spam*, etiquetados con un 1 si son *spam*, o con un 0 si no lo son (en cada uno de los 3 *datasets*); y otro modelo entrenado con un *dataset* conjunto que contiene *urls* seguras y *urls* de tipo *phishing*. De esta forma, la idea, ya mencionada, es que se analice el correo con ambos modelos para asegurar que no sea un correo de tipo *phishing*. He razonado que es mejor de esta forma, porque entrenar un solo modelo para textos de emails y textos de *urls* completamente distintos entre ellos, puede acabar siendo bastante complejo y dar resultados poco eficientes. La arquitectura de como entraría los *datasets* en el sistema de *machine learning* sería algo parecido a lo que se representa en la *Figura 18*.



Como se ve en la *Figura 18*, he decidido separar cada uno de los *datasets* en un 70% para datos de entrenamiento, y un 30% en datos de prueba. Cabe decir que he obtenido los *datasets* de emails *spam/ham*, y de URLs *phishing*, del sitio web *Kaggle* [33] [34]. En los *datasets* de emails, he encontrado y usado los tres *datasets* de emails *spam/ham* con dos columnas: una de ellas para para el

cuerpo del texto del email, y otra para la clase del email, pudiendo ser 1 o 0, con un total de 18650 filas entre los 3 *datasets* de emails. Por otro tengo un *dataset* de *urls* de tipo *phishing* y no *phishing*, con un total de 2 columnas correspondientes a las *urls* en sí, y una columna de etiquetado que indica si la URL es *phishing* o no *phishing*. De estas *urls* hay un total de 549346 filas.

De esta forma, de los *datasets* de emails *spam* obtengo 13055 para los datos de entrenamiento, y 5595 para datos de prueba. En cuanto a *urls*, obtengo un total de 384542 direcciones para datos de entrenamiento; y 164804 direcciones para datos de prueba.

## 3.2 Herramientas usadas

Las herramientas usadas para crear nuestro sistema para detección de phishing son, por un lado, el paquete de Python llamada *scikit-learn* (véase Anexo II), y por otro lado para el manejo de datos se usará *pandas* y *numpy*.

Por otro lado, también se usará el paquete de Python *imaplib*, que servirá para obtener los correos de mi cuenta de *Gmail* y almacenarlos en un editor de texto plano para que luego pueda proceder a analizar el contenido de dicho correo con los modelos creados.

Todo el proyecto será realizado en una interfaz especial para Python llamada *Anaconda*, que también analizaré.

### 3.2.1 Entorno de trabajo con *Anaconda*

*Anaconda* (en su versión 3, en el caso del proyecto realizado) es una distribución de los lenguajes de programación *Python* y *R* para computación científica, es decir, se usa para *data science*, *machine learning*, procesamiento de datos de gran escala, analíticas predictivas, etc..., que sirven para simplificar la gestión y despliegue de paquetes. La distribución *Anaconda* viene con más de 250 paquetes ya instalados, y permite instalar más de 7500 paquetes de código abierto adicionales desde *PyPI*, los paquetes del sistema *conda* (que es sistema de gestión de paquetes de *Anaconda*), o el gestor del entorno virtual. Incluye además una interfaz gráfica (*Anconda Navigator*), como una alternativa a la interfaz de línea de comandos *CLI*.

La principal diferencia entre *conda* y el gestor de paquetes *pip* es como se manejan las dependencias de los paquetes en cada uno. Cuando el gestor de paquetes *pip* instala un paquete, lo que hace es instalar automáticamente cualquier paquete de *Python* con dependencias sin comprobar si hay conflictos ya instalados con anterioridad. Esto puede afectar, por ejemplo, que, a la hora de trabajar con cierta librería importada, como *scikit-learn* por ejemplo, esta puede dejar de funcionar si hemos instalado con *pip* un paquete de *numpy* que requiere de una versión anterior o más actualizada de *scikit-learn*.

En contraste, *conda* analiza el entorno con el que se está trabajando actualmente, incluyendo todos los paquetes instalados, y, todo junto con una versión límite especificada, se organiza para instalar una serie dependencias compatibles entre sí, y en caso de que no pudiese hacerse tal cosa, el sistema de *conda* daría una señal de *warning*. Por eso *conda* existe para gente que trabaja dentro del campo de *data science* que necesita de librerías que funcionan en conjunto (como *numpy* con *pandas*), compatibles entre sí.

Junto a **Anaconda**, tenemos una interfaz gráfica de escritorio llamada *Anaconda Navigator*, que permite ejecutar varias aplicaciones y gestionar paquetes de *conda*, entornos y canales sin necesidad de hacer uso de la línea de comandos. Este navegador de *Anaconda* busca paquetes en *Anaconda Cloud* (para gestionar la paquetería en la nube propia de *Anaconda*), o en un repositorio local de *Anaconda*, los instala en el entorno de trabajo, ejecuta los paquetes, y los actualiza cuando sea necesario. Este navegador incluye las siguientes aplicaciones (de las cuales obviamente no vamos a usar todas), las cuales se describen a continuación:

- **JupyterLab**: Un entorno extensible para la computación interactiva y reproducible, basada en *Jupyter Notebook* y *Architecture*
- **Jupyter Notebook**: Entorno portátil de computación interactivo basado en web. Permite editar y ejecutar documentos legibles por humanos a la vez que describe el análisis de datos.
- **QtConsole**: Interfaz gráfica para usuarios de PyQt que admite figuras en línea, así como edición multilinea adecuada con resaltado de sintaxis, sugerencias gráficas, y mucho más
- **Spyder**: Entorno de desarrollo científico de Python. Además, es un potente IDE de Python con funciones avanzadas de edición, pruebas interactivas, depuración, e introspección.
- **Glue**: Visualización de datos multidimensionales en archivos. Explora las relaciones dentro y entre los conjuntos de datos relacionados.
- **Orange**: Framework de minería de datos basado en componentes. Visualización y análisis de datos para principiantes y expertos. Flujos de trabajo interactivos con una gran *toolbox*.
- **Rstudio**: Conjunto de herramientas integradas diseñadas para ayudar a ser más productivo con el lenguaje R. Incluye notebooks y todo lo básico para trabajar con R.
- **Visual Studio Code**: Editor de código optimizado con soporte para operaciones de desarrollo como depuración, ejecución de tareas, y control de versiones.

En el caso de mi proyecto he optado por usar *JupyterLab*, puesto que quería optar por una opción más visual, y este es una ampliación mejor y más potente y comprensible que *Jupyter Notebook*.

### 3.2.1.1 JupyterLab

*JupyterLab* [35] es una interfaz basada en web de la próxima generación construido para el proyecto *Jupyter*. Se puede vislumbrar una imagen de como se ve esta interfaz en *Google Chrome* en la *Figura 19*, donde a la izquierda se encuentra la jerarquía de archivos y directorios, y en el centro y la derecha de la pantalla el código.

Se puede ver, como ventaja, que en el que se puede ejecutar el código línea por línea sin necesidad de usar *breakpoints*. Esto hace que al final tanto los *inputs* como *outputs* sean más fáciles de ver, y al final acaba siendo un entorno más visual.

Como desventaja, es un editor de texto plano, por lo que al contrario que en otros entornos como *Visual Studio Code*, aquí el texto no se auto rellena.

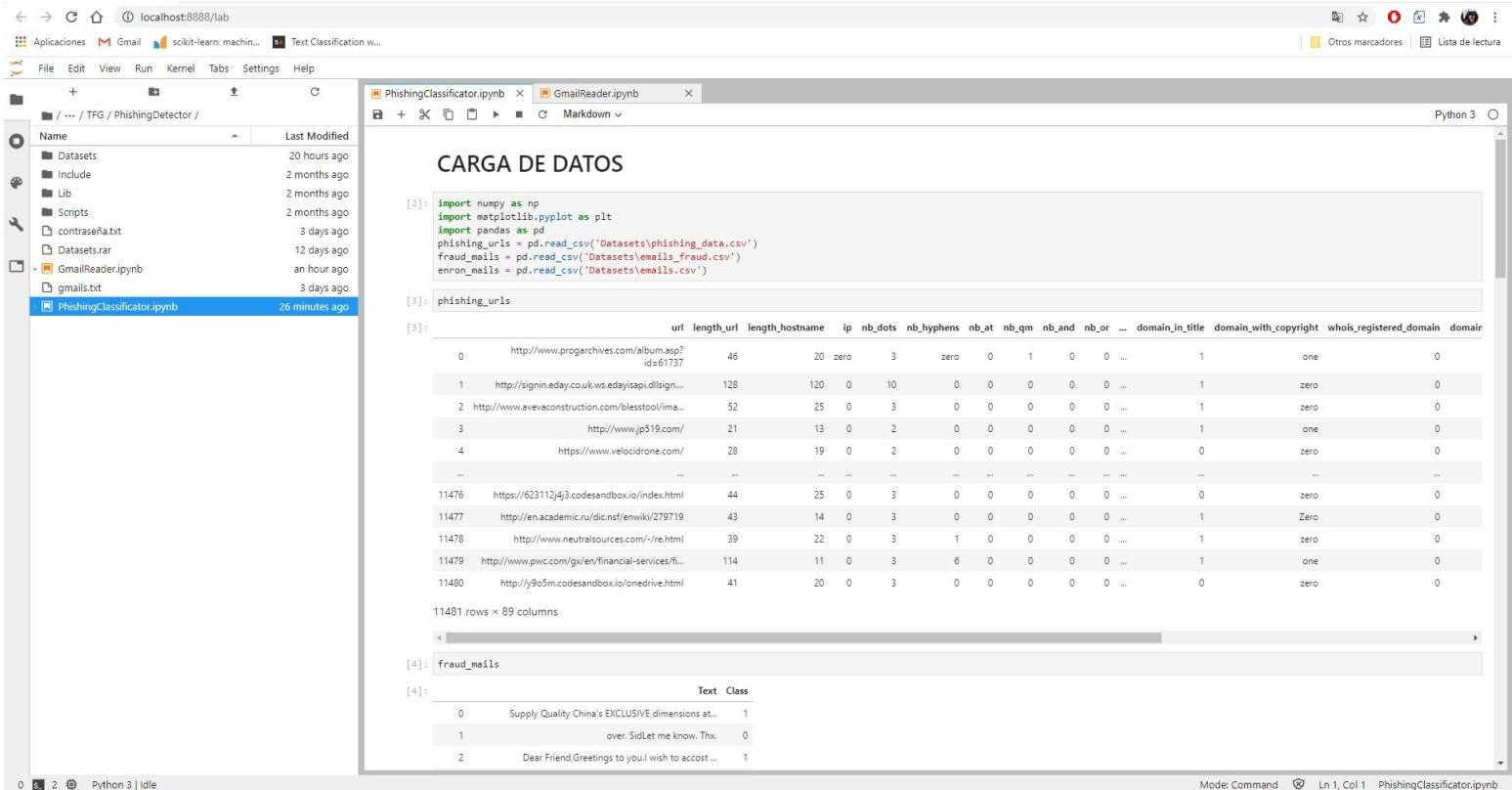


Figura 19: Interfaz visual de JupyterLabs con el código de mi proyecto

*JupyterLabs* permite trabajar con documentos y tareas como *Jupyter Notebook*, editores de texto, terminales, y componentes personalizables de un modo flexible, integrado, y extensible.

Se pueden organizar múltiples documentos y tareas de lado a lado en el entorno de trabajo usando pestañas y divisores. Los documentos y tareas se integran entre sí, activando nuevos flujos de trabajo para computación interactiva, como, por ejemplo:

- Las consolas de código proporcionan blocs de nota transitorias para código que se ejecuta interactivamente, con soporte total para unos *outputs* enriquecedores. Una consola de código puede ser enlazada con un *notebook kernel*.
- Los documentos respaldados por *kernel* permiten que el código de cualquier archivo de texto (*Markdown*, *Python*, *R*, *LaTeX*, etc...) se ejecute de forma interactiva en cualquier *kernel* de *Jupyter*.
- Los *outputs* de las celdas de un *notebook* se pueden duplicar en su propia pestaña, una al lado de otra, lo que permite paneles simples con controles interactivos respaldados por un *kernel*.
  - Varias vistas de documentos con diferentes editores o visores permiten la edición en vivo de documentos reflejados en otros visores.

*JupyterLab* también ofrece un modelo unificado para ver y manejar formatos de datos, y comprende muchos formatos de archivo (*JPEG* o *PNG*, *CSV*, *JSON*, *Markdown*, etc...).

Otra opción que ofrece *JupyterLab* es que, para navegar por la interfaz de usuario, se ofrecen atajos de teclado personalizables, y la capacidad de usar mapas de teclas de editores como *vim*, *Emacs*, o *Sublime Text*.

Por último, decir que *JupyterLabs* se sirve del mismo servidor, y utiliza el mismo formato de *notebook* que el *Jupyter Notebook* clásico.

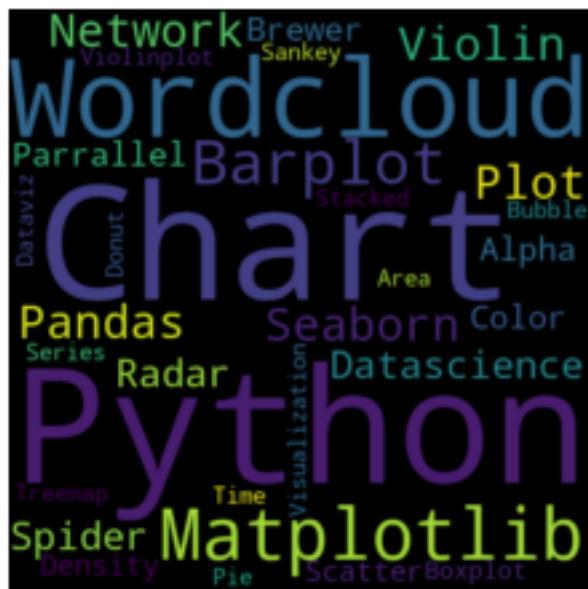
Dicho esto, voy a analizar las distintas librerías usadas en el código.

### 3.2.2 Librerías usadas en el código

Las principales librerías que hemos usado para el proyecto son:

- **numpy** [36]: Librería principal de *Python* para la informática científica, que proporciona potentes estructuras de datos, implementando matrices unidimensionales y multidimensionales. Estas estructuras de datos garantizan cálculos eficientes con matrices.
- **pandas** [37]: Librería de *Python* para el manejo y análisis de estructuras de datos.
- **matplotlib.pyplot** [38]: *matplotlib.pyplot* es una colección de funciones que hace que *matplotlib* funcione como *MATLAB*. Cada función *pyplot* realiza cambios a la representación de una figura, como puede ser la creación de dicha figura, crear un área de gráfico en la figura, dibujar líneas en el gráfico según la función establecida, decorar la gráfica con etiquetas de ejes, y muchas otras funciones.
- **nltk** [39]: *Natural Language ToolKit (NLTK)* es la biblioteca más popular en *Python* para el procesamiento del lenguaje natural, y tiene una comunidad muy grande detrás de ella. Es una herramienta fácil de aprender, hasta el punto de que es la la biblioteca de procesamiento más fácil de usar actualmente.
- **seaborn** [40]: *Seaborn* es una librería de visualización de datos construida en base a *matplotlib*, e integrada con las estructuras de datos de *pandas* en *Python*. La visualización es la parte central de la librería *seaborn*, que ayuda en la exploración y comprensión de los datos.
- **wordcloud** [41]: *wordcloud* (también llamado nube de etiquetas o lista ponderada) es una representación visual de datos textuales. Lo que hace esta librería básicamente es obtener las palabras individuales de un texto, y la importancia de cada palabra es representada en una figura con un color y tamaño de fuente dependiente de la importancia de dicha palabra (si es más frecuente en el texto, por ejemplo). Un ejemplo de la figura representada se muestra en la *Figura 20*.
- **scikit-learn** [42]: Esta librería va a ser el eje central en el que se van a implementar los algoritmos de *machine learning*. Para una explicación más detallada, ir al anexo.
- **imaplib**: Esta librería permite leer los correos de una cuenta de *Gmail* personal.
- **email** [43]: El paquete *email* de *Python* es una librería que maneja los mensajes de email, incluidas las parte *MIME* y otros documentos de mensajes basados en *RFC-2822*. Sin embargo, esta librería no ha sido diseñada para realizar ningún tipo de envío de mensajes email a servidores *SMTP*, *NNTP*, u otros servidores; para esto se utilizarían otras funciones de modulo como *smtplib* o *nntplib*.
- **pickle** [44] [45]: El módulo *pickle* de *Python* se usa principalmente para serializar y des serializar objetos estructurados en *Python*. En otras palabras, es el proceso de convertir un objeto de *Python* en un *stream* de bytes para almacenarlos en un fichero o base de datos, mantener el programa entre sesiones, o para transportar los datos a través de la red. Tiene un cierto parecido con el módulo *json*, aunque difiere de este en

que serializa los objetos en un formato binario, lo que significa que el resultado no es leible para los humanos.



*Figura 20: Ejemplo de la figura representada en un texto por wordcloud, con las palabras más frecuentes de dicho texto*

Mencionadas y explicadas las librerías usadas en el código, ahora voy a proceder a explicar la implementación del código realizado para el proyecto con las herramientas explicadas.

### **3.3 Implementación del código**

Una vez explicados el planteamiento arquitectico del proyecto, y cuáles van a ser las herramientas usadas, voy a entrar de lleno a explicar el código realizado para poder cumplir los objetivos del proyecto en relación al entrenamiento y testeo del sistema de *machine learning* que pretendo crear. Para ello, voy a detallar las distintas partes en las que se compone el proceso de carga, preparación o preprocesamiento, y otras fases previas a poder finalmente entrenar los datos con los algoritmos seleccionados, como se va a ver en los siguientes subapartados.

### **3.3.1 Clasificación de emails y urls**

En las primeras dos clases que voy a explicar, lo que voy a hacer es cargar los datasets conseguidos de *Kaggle*, los tres datasets de emails [33] para la clasificación de emails, y el dataset de urls [34] para la clase que clasifica las urls. A pesar de ser clases distintas, he decidido explicarlas en conjunto, ya que en lo único que difieren ambas clases es en el número de datasets y el tipo de datasets de cada clase, pero en lo que se refiere al preprocesamiento de datos, extracción de características, o entrenamiento de algoritmos, tienen el mismo procedimiento. En la clase de clasificación de emails, cada uno de los tres datasets contienen varios correos de tipo *spam* () o *ham* (), etiquetados los correspondientes correos en su columna con 1 si son *spam*, o 0 si son de tipo *ham*. En la clase de clasificación de urls, cada una de las urls tiene un etiquetado que indica si esa url es de tipo *phishing* si está marcada con la cadena de caracteres *bad*, o de tipo legítimo, si está marcada con la cadena de caracteres *good* (que luego se convertirán en 1 y 0 durante el preprocesamiento de los

datos). Tras la carga de datos, haré un preprocesamiento de los datos (eliminando columnas innecesarias, o añadiendo otras), y luego extraeré con los métodos adecuados ciertas características interesantes de los emails y *urls*, que se analizarán. Por último, separaré en datos de entrenamiento y de prueba los datos, y los entrenaré con los modelos seleccionados, para finalmente mostrar los resultados obtenidos y las métricas, y seleccionar cuál es el mejor método de clasificación. Pero antes de esto, primero vamos al principio, a la carga de datos.

### 3.3.1.1 Carga de datos

En este primer punto, como bien he dicho, voy a cargar los *datasets* de los CSV descargados, tanto los de mails, como los de *urls*, con el método *read\_csv* de Python.

Como ya mencioné con anterioridad, se van a cargar tres *datasets* de emails con distintos corpus, etiquetados los tres con 1 si son *spam* o con 0 si sin *ham*. Los tres *datasets* se muestran en las tablas correspondientes a la *Tabla 17*, la *Tabla 18*, y la *Tabla 19*.

		<b>Cuerpo</b>	<b>Etiqueta</b>
0		\nSave up to 70% on Life Insurance.\nWhy Spend...	1
1		1) Fight The Risk of Cancer!\nhttp://www.adcli...	1
2		1) Fight The Risk of Cancer!\nhttp://www.adcli...	1
3		#####	1
4		I thought you might like these:\n1) Slim Down ...	1
...		...	...
6041		empty	0
6042		— — ...	0
6043		IN THIS ISSUE:01. Readers write\n02. Extension...	0
6044		empty	0
6045		empty	0

*Tabla 17: Primer dataset de emails spam/ham*

		<b>Cuerpo</b>	<b>Etiqueta</b>
0		Subject: stock promo mover : cwtd\n * * * urge...	1
1		Subject: are you listed in major search engine...	1
2		Subject: important information thu , 30 jun 20...	1
3		Subject: = ? utf - 8 ? q ? bask your life with...	1
4		Subject: " bidstogo " is places to go , things...	1
...		...	...
9995		Subject: monday 22 nd oct\n louise ,\n do you ...	0
9996		Subject: missing bloomberg deals\n stephanie -...	0
9997		Subject: eops salary survey questionnaire\n we...	0
9998		Subject: q 3 comparison\n hi louise ,\n i have...	0
9999		Subject: confidential folder to safely pass in...	0

Tabla 18: Segundo dataset de emails spam/ham

		Cuerpo	Etiqueta
0		Subject: great part-time or summer job !\n \n ...	1
1		Subject: auto insurance rates too high ?\n \n ...	1
2		Subject: do want the best and economical hunti...	1
3		Subject: email 57 million people for \$ 99\n \n ...	1
4		Subject: do n't miss these !\n \n attention ! ...	1
...		...	...
2600		Subject: computationally - intensive methods i...	0
2601		Subject: books : a survey of american linguist...	0
2602		Subject: wecol ' 98 - - western conference on ...	0
2603		Subject: euralex ' 98 - revised programme\n \n...	0
2604		Body,Label\n 0,"Subject: great part-time or s...	0

Tabla 19: Tercer dataset de emails spam/ham

Como se puede observar, todos los *datasets* están planteados en formato *Unicode* [46], el cual es el formato estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de numerosos idiomas y disciplinas técnicas. Así, los saltos de línea están representados con *\n* o los tabulados con *\t*. Más adelante, en el preprocessamiento de datos pretendo limpiar todos los mensajes para que este tipo de caracteres molesto como puedan ser los ya mencionados saltos de líneas o tabulados, desaparezcan, y el texto quede como un texto lineal, normal y corriente.

También hay que tener en cuenta que en los *datasets* de la *Tabla 18* y la *Tabla 19*, aparte del cuerpo del mensaje como tal, tienen la parte del asunto del mensaje, o *subject*, que para esta implementación eliminaremos en la etapa del preprocessamiento de datos.

Finalmente, una cosa que también he contemplado analizar es cuantos datos de correos *spam* y *ham* hay en total en cada uno de los tres *datasets*, para ver si los datos están relativamente balanceados. Los resultado obtenidos se muestran en las figuras *Figura 21*, *Figura 22*, y *Figura 23*.

```
<matplotlib.axes._subplots.AxesSubplot at 0x27b015e6ec8>
```

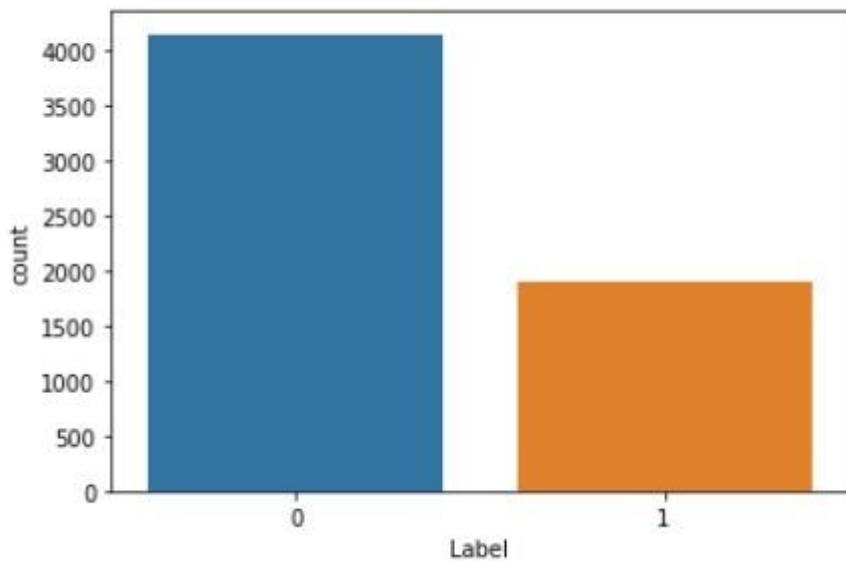


Figura 21: Grafico que muestra el balance de datos entre los emails ham (0) y los emails spam (1) del primer dataset de emails de la Tabla 17

```
<matplotlib.axes._subplots.AxesSubplot at 0x27b01776e08>
```

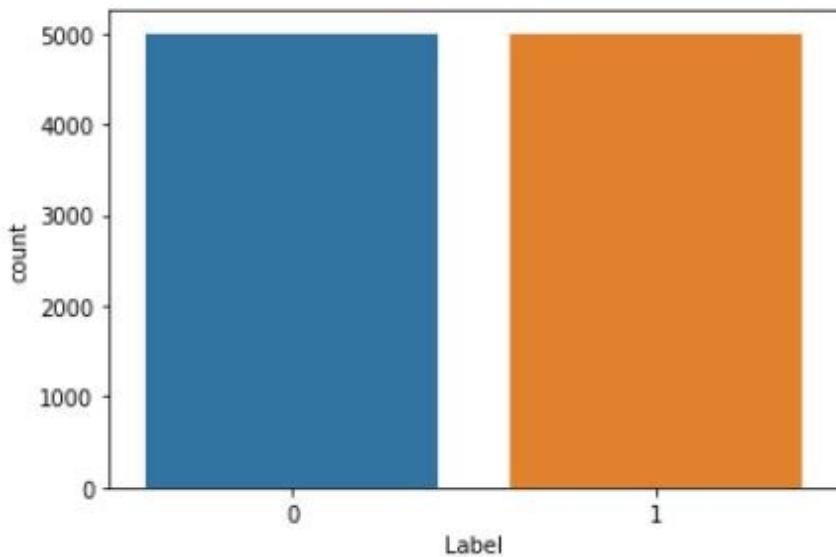


Figura 22: Grafico que muestra el balance de datos entre los emails ham (0) y los emails spam (1) del primer dataset de emails de la Tabla 18

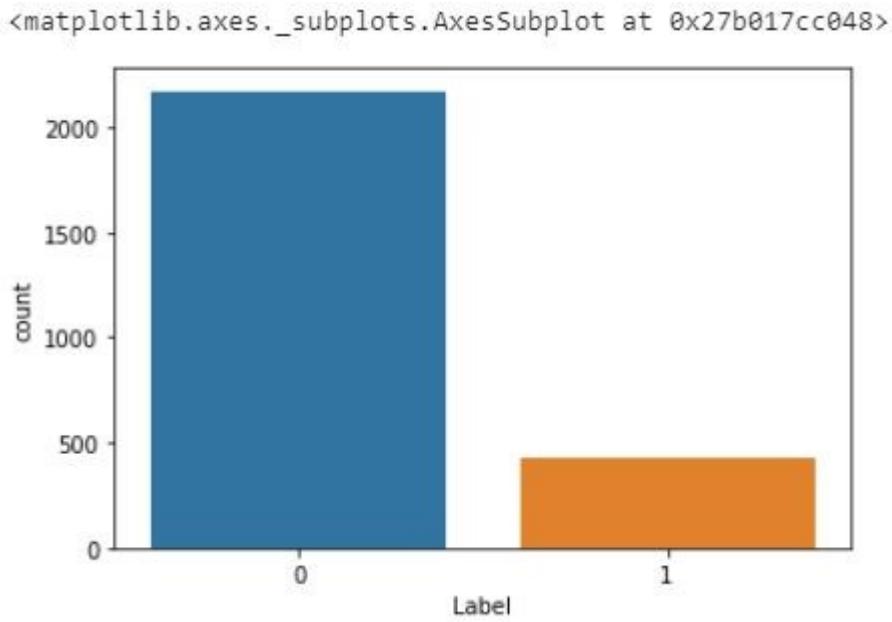


Figura 23: Grafico que muestra el balance de datos entre los emails ham (0) y los emails spam (1) del primer dataset de emails de la Tabla 19

Como se puede observar, existe una cantidad mayor de datos *ham* que de *spam*, por lo que los datos están un poco desbalanceados. El equilibrio de ambos tipos de datos se conseguirá más adelante, en las siguientes etapas de código. Además, también se juntarán todos los emails en un solo *DataFrame* para tenerlos más claros y ordenados. Un *DataFrame* [47] es una estructura de datos con dos dimensiones en la cual se puede guardar datos de distintos tipos (como caracteres, enteros, valores de punto flotante, factores y más) en columnas.

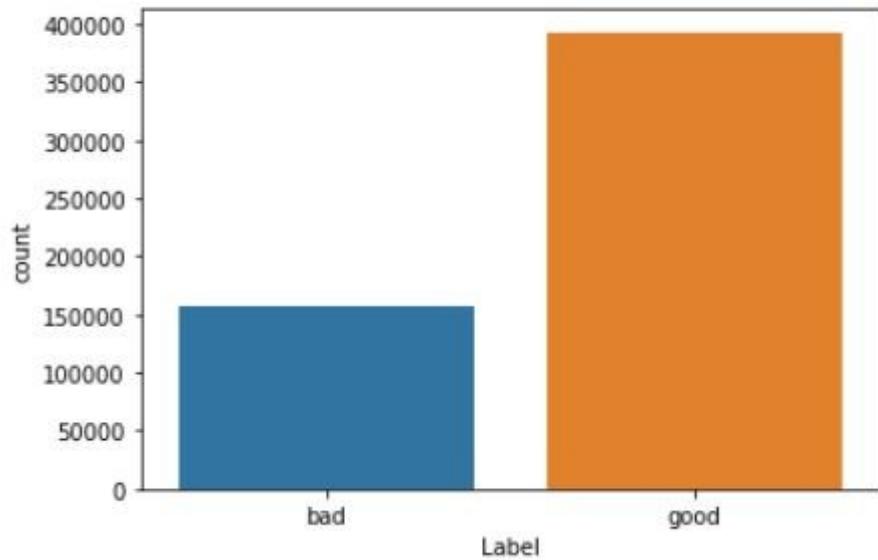
Vamos ahora a observar la otra clase, la de *urls*. En esta clase, solo existe un *dataset*, que la cargarlo igual que con los emails (con *read\_csv*), se obtiene el *DataFrame* mostrado en la Tabla 20. En este *DataFrame*, se etiquetan las *urls* de tipo *phishing* como *bad*, y las que no son de tipo *phishing*, con la etiqueta *good*.

		<b>URL</b>	<b>Etiqueta</b>
0		nobell.it/70ffb52d079109dca5664cce6f317373782/...	bad
1		www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscr...	bad
2		serviciosbys.com/paypal.cgi.bin.get-into.herf....	bad
3		mail.printakid.com/www.online.americanexpress....	bad
4		thewhiskeydregs.com/wp-content/themes/widescre...	bad
...		...	...
549341		23.227.196.215/	bad
549342		apple-checker.org/	bad
549343		apple-iclod.org/	bad
549344		apple-uptoday.org/	bad
549345		apple-search.info	bad

Tabla 20: Dataset correspondiente a las urls de tipo phishing/legítimo

Una vez mostrado el *DataFrame*, voy a proceder a analizar la distribución de datos, que se muestra en la gráfica de la *Figura 24*.

```
<matplotlib.axes._subplots.AxesSubplot at 0x24530c0eb08>
```



*Figura 24: Grafica que muestra el total de datos de urls etiquetados como phishing (bad) o como no phishing (good)*

Como se puede observar, igual que en los emails, los datos estás desproporcionados, teniendo un número mayor de *urls* de tipo *phishing*, que de tipo no *phishing*. Este desequilibrio, de nuevo, se corregirá en etapas de código posteriores.

### 3.3.1.2 Preprocesamiento de datos

Como he mencionado en el capítulo anterior, lo primero que voy a realizar tiene que ver con los *dataset* de emails *spam/ham*. He creado tres *DataFrame* separados para cada uno, y luego los he concatenado.

Para tener solo un *dataset* en vez de tres separados, realizó una concatenación y cambio en los nombres de las columnas de los tres emails, usando el método de *pandas pd.concat*, para crear un único *dataset* más conciso y concreto, que quedaría con un total de 18650 emails. Un esbozo de este nuevo *DataFrame* creado con la concatenación de los tres *DataFrames* se muestra en la *Tabla 21*.

		Cuerpo	Etiqueta
0		\nSave up to 70% on Life Insurance.\nWhy Spend...	1
1		1) Fight The Risk of Cancer!\nhttp://www.adcli...	1
2		1) Fight The Risk of Cancer!\nhttp://www.adcli...	1
3		#####	1
4		I thought you might like these:\n1) Slim Down ...	1
...		...	...
18646		Subject: computationally - intensive methods i...	0
18647		Subject: books : a survey of american linguist...	0
18648		Subject: wecol '98 -- western conference on ...	0

18649	Subject: euralex ' 98 - revised programme\n \n...	0
18650	,Body,Label\n 0,"Subject: great part-time or s...	0

Tabla 21: DataFrame que se corresponde a la concatenación de los tres datasets de emails en uno solo

Luego de esto, tanto en la clase de clasificación de emails, como en la de *urls*, elimino todos los correos y *urls* que estén repetidos (con el método *drop\_duplicates*), y los que tengo un valor nulo o *None* en Python (con el método *dropna*), dejándome esto con un total de 18650 correo y 507196 *urls*. Haciendo ahora, tras este proceso de concatenación y eliminación de correos innecesarios, de la comparativa de correos *spam* y *ham*, obtengo la gráfica que se muestra en la Figura 25.

```
<matplotlib.axes._subplots.AxesSubplot at 0x27b8ec400c8>
```

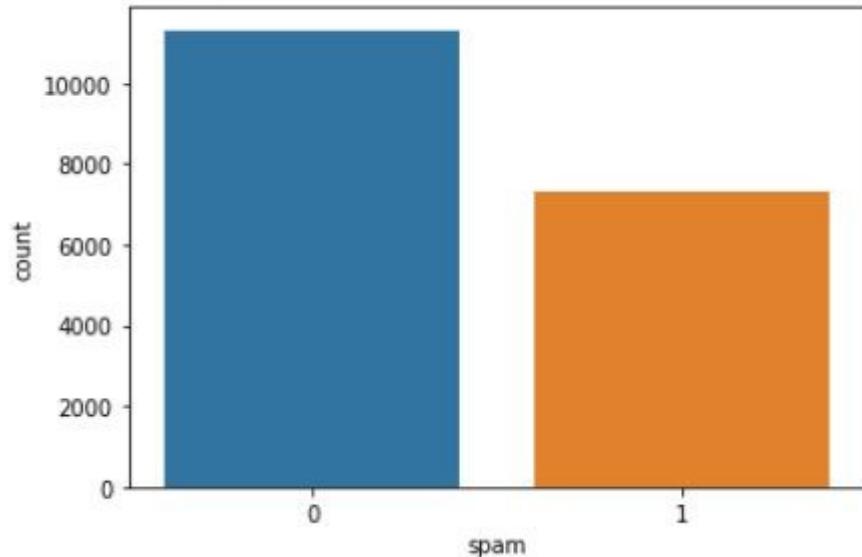


Figura 25: Grafico comparativo que muestra la cantidad de datos usados como *ham* (0) y como *spam* (1) de la concatenación de emails

En cuanto a lo que se refiere a *urls*, he eliminado las *urls* duplicadas y de valor nulo [48], y he hecho un mapeado de la categoría de *Etiquetas* dentro de los *datasets* de *urls*, de modo que para hacerlo más tangible de entender para los algoritmos de *machine learning* más adelante, he cambiado la etiqueta *bad* por 1, y la etiqueta *good* por 0. Gráficamente, esta nueva comparación entre *urls phishing* y no *phishing* es la que se muestra en la Figura 26.

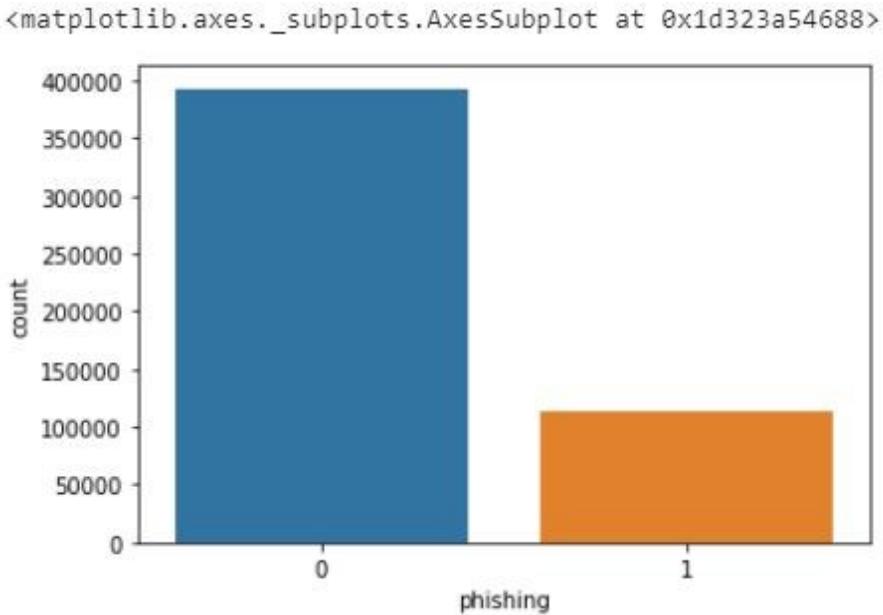


Figura 26: Grafica que muestra el total de datos de urls etiquetados como phising (bad) o como no phising (good) tras eliminar los datos innecesarios y mapear la columna de Etiquetas

Como se puede observar, los datos están bastante desbalanceado, por lo que voy a realizar a continuación es balancear los datasets [49]. Consigo esto con la clase *resample* [50], separando los *DataFrames* en aquellos que sean la clase mayoritaria (los emails de tipo *ham*), y la clase minoritaria (los emails *spam*). Una vez hecho esto, con la clase *resample*, se pasan como parámetros el conjunto de datasets de la clase minoritaria que se quiere remodelar, y el número de muestras que se quiere generar para que se balancee con respecto a los emails de la clase mayoritaria *ham*. Después de esto, se vuelven a concatenar ambos datasets de emails. El código que muestra esta remodelación para alcanzar el balanceado de datos se muestra en la Tabla 22. Cabe decir que el mismo código sirve para la clase de *urls* entre las *urls phsiing* y *no phising*, con el *dataset* correspondiente de dicha clase.

```
from sklearn.utils import resample

df_majority = emails[emails.spam==0]
df_minority = emails[emails.spam==1]

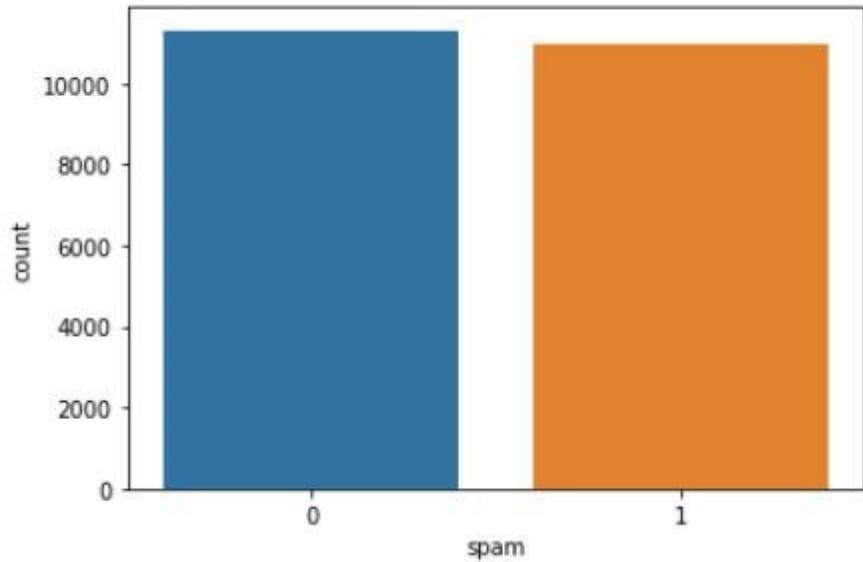
# muestreo de la clase minoritaria
df_minority_upsampled = resample(df_minority,
                                   replace=True,      # muestras con reemplazamiento
                                   n_samples=11000,
                                   random_state=42) # resultados reproducibles

# concatenación de Los emails ham con Los spams tras la remodelación
emails = pd.concat([df_majority, df_minority_upsampled])
```

Tabla 22: Código creado para obtener un balanceamiento de datos en el dataset de emails

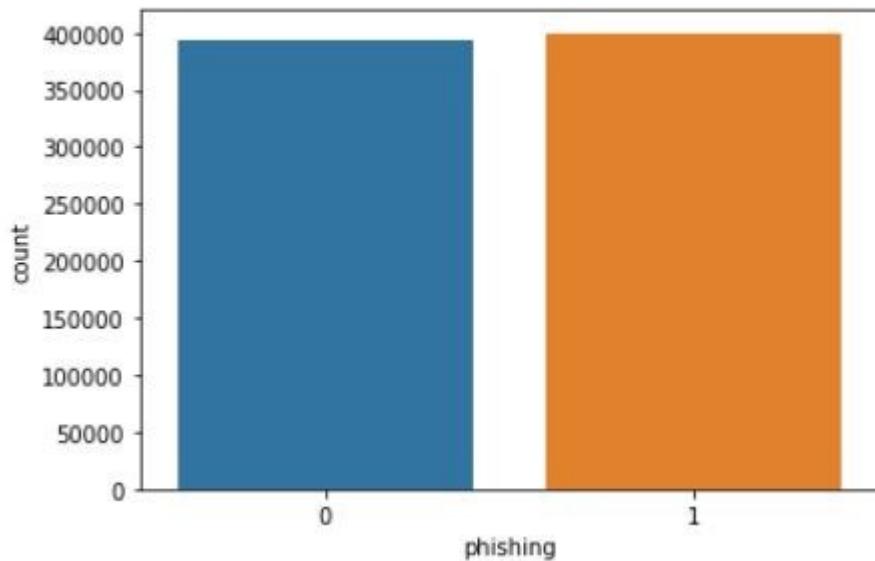
Una vez hecho esto, las gráficas de los datos con el balance conseguido se muestran en la *Figura 27* para los emails, y en la *Figura 28* para las *urls*.

```
<matplotlib.axes._subplots.AxesSubplot at 0x27b908b4e08>
```



*Figura 27: Gráfica con los datos de emails balanceados*

```
<matplotlib.axes._subplots.AxesSubplot at 0x22ac9cbbc88>
```



*Figura 28: Gráfica con los datos de urls balanceados*

Una vez balanceados los datos, lo siguiente que voy a hacer es limpiarlos de palabras o símbolos incensarios, como puedan ser los saltos de línea o tabulados, los caracteres especiales (&, |, (), [], {}, etc...), o dígitos, o las llamadas en inglés *stopwords*, que son palabras que no aportan nada al análisis léxico, como puedan ser preposiciones o palabras que conecten unas con otras, como *the, of, from, before, after*, etc...

Además, también voy a tokenizar o normalizar el texto [51]. Esto es el proceso de transformación de texto para la obtención de una forma canónica.

Usualmente incluye la corrección de errores ortográficos y de tipeo, y la uniformización en el uso de mayúsculas, acentos, signos de puntuación, acrónimos y abreviaciones.

Por último, se realiza una lematización [52] del texto. Esto es el proceso de convertir una palabra en su forma base, o dicho de otro modo, el proceso de agrupar las formas flexionadas de una palabra para que puedan ser analizadas léxicamente como un solo objeto, identificadas como el lexema de una palabra.

Después de esto, simplemente construyo un vector con todos los correos tras esta limpieza, para que se puedan transformar y pasar correctamente en el formato que piden los algoritmos de *machine learning*. Toda esta parte se puede contemplar en el código de la *Tabla 23*.

Sin embargo, para las *uls*, el proceso de limpieza es distinto, y el código correspondiente a su limpieza viene representado en la *Tabla 24*. En este fragmento de código lo que se hace es, primeramente, igual que se hizo con los emails, tokenizar o normalizar el texto, esta vez con un mapeado de las palabras. Luego, he procedido a usar la función *SnowballStemmer*, la cual realiza un procedimiento con el texto pasado de las palabras normalizadas, llamado *stemming* [53], que un método para reducir una palabra a su raíz o a un *stem* del inglés, y que sirve para aumentar la métrica *recall*, que es una medida que se explicará mejor en el capítulo de *Análisis de resultados obtenidos*.

Hecho esto, simplemente se mapea el texto con las palabras raíz obtenidas tras el proceso *stemming*, y se usa como la *url* normalizada y clara que se transformará para poder pasarela al algoritmo de *machine learning*.

```
# Limpieza de datos para Los correos
corpus = []
lemmatizer = WordNetLemmatizer()

for email in mails['Email']:
    # eliminar el asunto del mensaje, Los tabulados, y Los saltos de Línea
    removed_tabs_newline = re.sub('[\n|\t]', ' ', email)
    removed_subject = re.sub('Subject:', ' ', removed_tabs_newline)

    # eliminar Los caracteres especiales y Los dígitos
    removed_spchar_digits = re.sub('[^a-zA-Z]', ' ', removed_subject)

    # transformar Los emails a minusculas
    lower_case_email = removed_spchar_digits.lower()

    # normalizar Los emails por palabras o separar por palabras
    tokenized_email = lower_case_email.split()

    # eliminar Las stopwords
    filtered_words = [word for word in tokenized_email if word not in stopwords.words('english')]
```

```

# Lematización de Las palabras
lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words]

# contruir el corpus del email con el texto Limpio
email = ' '.join(lemmatized_words)
corpus_fraudulent.append(email)

```

Tabla 23: Código para limpiar los datos de los emails

```

# Limpieza de datos para las urls
tokenizer = RegexpTokenizer(r'[A-Za-z]+')
phishing_urls.loc[:, 'text_tokenized'] = phishing_urls.url.map(lambda t: tokenizer.tokenize(t))
root_words = SnowballStemmer("english")
phishing_urls.loc[:, 'root_words'] = phishing_urls['text_tokenized'].map(
    lambda l: [root_words.stem(word) for word in l])
phishing_urls.loc[:, 'text_sent'] = phishing_urls['root_words'].map(lambda l: ' '.join(l))

```

Tabla 24: Código para analizar y limpiar las palabras dentro de las urls

Una vez realizado todo el proceso de preprocessamiento de datos, voy a proceder a la etapa de extracción de características, donde se analizarán las palabras más recurrentes en los emails *spam* y *ham* de los textos limpios, de igual manera que las palabras más usadas en las *urls* de tipo *phishing* y no *phishing*.

### 3.3.1.3 Extracción de características

Lo primero que voy a realizar en cuanto a extracción de características es analizar las palabras más usadas en los correos de tipo *ham* y *spam*, y de igual modo, en las *urls* de tipo *phishing* y de tipo *no phishing*.

Para hacer esto, he usado la librería *Wordcloud*, explicada en el capítulo de *Librerías usadas en el código*. Usando los textos, tanto de emails como de *urls* limpios, y añadiendo las *stopwords* como palabras a obviar que no se deberían tener en cuenta, creo el diagrama con las palabras más recurrentes en cada tipo de email/*url*.

En la *Figura 29* se muestran las palabras más frecuentemente usadas en los correos de tipo *ham*; en la *Figura 30* se muestran las palabras más frecuentemente usadas en los correos de tipo *spam*; en la *Figura 31* se muestran las palabras más frecuentemente usadas en las *urls* que no contienen *phishing*; y en la *Figura 32* se muestran las palabras más frecuentemente usadas en los las *urls* con contenido *phishing*.

## Palabras más usadas en Mails de tipo Ham



Figura 29: Diagrama de las palabras más usadas en los emails de tipo ham

Como se puede observar, en los emails *ham*, o dicho de otro modo, emails legítimos, algunas de las palabras destacadas que más se repiten son *enron*, *language*, *university*, *information*, *linguistic*, o *paper*. La primera de ellas, *enron*, es una palabra muy repetida debido a que muchos de los correos de tipo *ham* usados para este proyecto en los datasets elegidos, se han sacado de un dataset muy grande de correos de una empresa ficticia llamada *Enron* [54], los cuales son todos correos creados artificialmente como si pertenesen a esta empresa, pero siendo todos ellos de tipo legítimo. Otras palabras también acuñadas a ser de emails de empresa se pueden ver, como pueden ser *company*, *workshop*, *conference*, *service*, *work*, o incluso *information*. Este tipo de emails *ham* también pueden ser de ámbito educativo, como se muestra en palabras como *language*, *university*, *education*, o *research*. Al final, aunque haya alguna palabra que se pueda corresponder más con un email personal, la gran mayoría de emails seleccionados de los datasets son emails empresariales o educativos, donde puede haber más posibilidad de posible contenido fraudulento.

## Palabras más usadas en Mails de tipo Spam



Figura 30: Diagrama de las palabras más usadas en los emails de tipo spam

Como se puede observar, algunas de las palabras más usadas en los emails de tipo *spam* con posibilidad de tener contenido dañino, son palabras como *click*, *free*, *money*, *business*, *price*, o *address*. Cuando dije antes que los emails de tipo *spam* solían tener contenido empresarial o educativo, no me equivocaba, pues también aquí se repiten algunas palabras propias de los emails *ham*, como pueden ser *company*, *report*, o *service*. Esto se debe obviamente a que un empleado dentro de una empresa se podría fiar de este tipo de emails empresariales, igual que un estudiante de emails educativos. Una cosa que llama mucho la atención es la palabra *click*, pues esto indica que muchos de los emails *spam* pueden contener esta palabra porque también incluyen un enlace, como “Haga *click* aquí” para acceder a un sitio web determinado, que puede ser obviamente engañoso, es decir, esto sería un email de tipo *phishing*. También otra palabra muy destacada señal de que el correo puede ser engañoso es la palabra *free*, la cual puede estar indicando que muchos de estos correos tipo *spam* pueden prometer cierto producto o servicio gratis para engañar al usuario que abre el correo, y así llevarle a una página con contenido dañino, por ejemplo.

Dicho esto, voy a hablar ahora de las palabras más frecuentes en las *urls* de tipo *phishing* y de tipo no *phishing*.

## Palabras más usadas enUrls de tipo no Phishing



Figura 31: Diagrama de las palabras más usadas en las urls de tipo no phishing

En las *urls* que no tienen contenido *phishing* se puede observar que algunas de las palabras más recurrentes y destacadas son *index*, *html*, o *dtype*, entre otros. Se puede decir que *index* [55] indica una URL o archivo local que se carga automáticamente cuando se inicia un navegador web y cuando se presiona el botón "Inicio" del navegador. El término también se utiliza para referirse a la página principal, el índice del directorio del servidor web o la página web principal de un sitio web de un grupo, empresa, organización o individuo. Por lo tanto, el uso de *index* es sinónimo de que se está accediendo a un sitio web seguro, y de igual manera sucede con las páginas web en formato *html* (*Hypertext Markup Language* o Lenguaje de Margen de Hipertexto) [56], siendo este el código que se suele usar para estructurar una página web y su contenido. Finalmente, *dtype* también puede ser señal de que se está accediendo a una página web segura, puesto que indica cómo se deben interpretar los bytes en el bloque de memoria de tamaño fijo correspondiente a un elemento de un objeto concreto.

Vamos ahora a observar las palabras más frecuentes en las *urls* de tipo *phishing* para contrastar.

## Palabras más usadas en URLs de tipo Phishing



Figura 32: Diagrama de las palabras más usadas en las urls de tipo phishing

Se puede observar que algunas de las palabras más destacadas en las *urls* de tipo *phishing* son *ly*, *content*, *imag*, o *karma*. Algunas de estas como *karma* o *imag*, son páginas webs de tipo engañoso que suelen usar la palabra específica, pues buscando, he encontrado varias webs que piden datos personales, como la compra-venta de productos de lujo como joyas, o por ejemplo páginas webs bancarias que piden datos delicados acerca de la tarjeta de crédito para realizar una estafa.

Otros sitios webs usan terminaciones específicas que suelen llevar a sitios webs fraudulentos, como puede ser *.ly*, o, mejor dicho, *.ly* [57]. Este por ejemplo es el proceso de registrar un nombre de dominio de usuario dentro del dominio de nivel superior con código de país (*ccTLD*) para Libia, que puede parecer seguro, pero al ser un dominio poco usado y de libre mercado, lo usan muchos *hackers* para sus planificaciones de ataque sobre un conjunto de individuos, como pueda ser una empresa.

Dicho esto, lo siguiente que voy a hacer es transformar los textos, tanto de los emails, como de las *urls*, para que se puedan pasar como argumentos a los algoritmos de *machine learning*.

Lo primero que voy a hacer es transformar los emails. Para ello voy a usar la función *TfidfVectorizer* y *tfidf.fit\_transform*, las cuales convierten una colección de textos en bruto, en una matriz de características *TF-IDF* [58]. Esta matriz *TF-IDF* se puede describir como una matriz de pesos numéricos, que miden la importancia que tiene cada palabra dentro del texto o *corpus*, y tiene mayor o menor peso dependiendo de esa importancia. Así, por ejemplo, observando las palabras más frecuentes de los emails *spam* en la *Figura 30*, tendría un mayor peso una palabra como *money* o *company*, que, por ejemplo, *click*.

Luego, con la función de *tfidf.get\_feature\_names*, se mapean las palabras usadas para guardarlas como características del texto. Así, para la que he llamado la variable *X*, que es el *input* para los algoritmos de *machine learning*, he creado un *DataFrame* en el que las filas son los vectores con los pesos de cada una de las palabras, y las columnas las palabras en cuestión, pasadas como características del texto. Sin embargo, la variable *y* u *output* para los algoritmos de *machine learning* usados, corresponde a la columna *spam* del *DataFrame* de emails concatenados creado, en el que numéricamente se marca un 1 si el email es de tipo *spam*, o 0 si es de tipo *ham*. El código correspondiente se puede observar en la *Tabla 25*, y un ejemplo de la matriz *X* creada con las palabras y sus respectivos pesos se puede observar en la *Figura 33*.

```
# creación de los vectores usando TF-IDF

tfidf = TfidfVectorizer(max_features=5000)
vectors = tfidf.fit_transform(corpus).toarray()
feature_names = tfidf.get_feature_names()

# extracción de las variables independientes y dependientes del dataset
X = pd.DataFrame(vectors, columns=feature_names)
y = emails['spam']
```

*Tabla 25: Ejemplo de transformación de los emails spam con el vector de valores TF-IDF para pasárselo como argumento al modelo de machine learning*

	aa	aaa	ab	abacha	ability	able	abroad	absence	absolute	\
0	0.0	0.0	0.0	0.0	0.0	0.047933	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
22317	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
22318	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
22319	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
22320	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
22321	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	

*Figura 33: Matriz TF-IDF resultado de la transformación con los textos de los emails*

Para las *urls* sin embargo, se ha usado otro método para la transformación: el llamado *CountVectorizer* [59], debido a que es más complicado desde un principio separar y calcular los pesos de las distintas palabras. Esta función se utiliza para transformar un texto dado en un vector basado en la frecuencia de cada palabra que hay en el texto. Para ellos, uso la función *fit\_transform* de esta

librería de *CountVectorizer*. El código se puede observar en la *Tabla 26*, y un esbozo del vector de frecuencias que se crearía se puede observar en la *Tabla 27*. Luego, lo que hago es pasar como variable *input X* este vector creado con *CountVectorizer*, y como *output y* paso la columna del *DataFrame* de *urls* que identifica si la *url* es de tipo *phishing* (con 1) o no (con 0).

```
c = CountVectorizer()
cv = c.fit_transform(phishing_urls.text_sent)

# extracción de las variables independientes y dependientes del dataset
X = cv
y = phishing_urls ['phishing']
```

*Tabla 26: Ejemplo de transformación de las urls con el vector de valores CountVectorizer para pasárselo como argumento al modelo de machine learning*

	<b>esxcc</b>	<b>com</b>	<b>js</b>	<b>index</b>	<b>htm</b>	<b>us</b>	<b>battl</b>	<b>net</b>	<b>noghn</b>	<b>en</b>
<i>phishing</i>	0	0	0	0	0	0	0	0	0	0
<i>No phishing</i>	94601	59428	156588	142243	135955	3312201	25365	207977	212454	91494

*Tabla 27: Esbozo del vector de frecuencia de las 10 primeras palabras de las urls creado con CountVectorizer*

Una vez transformados los datos, la sección siguiente de código va a ser la más importante sin duda, que es entrar en lo que se refiere a la creación de datos de entrenamiento y de prueba, así como la creación de los algoritmos de entrenamiento, y el uso de los datos para entrenarlos y probarlos.

### 3.3.1.4 Entrenamiento de algoritmos

En esta fase, como bien he dicho, voy a entrenar a los modelos con los algoritmos seleccionados. Para ello, lo primero que voy a hacer es usar la función *train\_test\_split*, como se muestra en el código de la *Tabla 28*. Esta función coge nuestras variables *X* e *y* correspondientes a los *inputs* y *outputs* que se pasa al algoritmo de *machine learning* correspondiente, establecidas en la etapa de preprocesamiento de datos, como argumentos, así como el tamaño de los datos de prueba, que se corresponden, como bien se dijo en la sección de *Datasets*, a un 30% de los datos totales, y un 70% para datos de entrenamiento (el cálculo se hace automático con la función de *train\_split\_set* pasándole solo uno de los argumentos de tamaño de datos de prueba/entrenamiento).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)
```

*Tabla 28: Código correspondiente a la separación de los datos en los datos de entrenamiento y los datos de prueba*

Una vez hecha esta separación, se va a proceder a crear los distintos modelos. En el código de la *Tabla 29* se muestra como construir un modelo, con el algoritmo de Bayes ingenuo. Con mostrar solo un ejemplo vale para todos, pues, aunque los distintos algoritmos funcionen de distinta forma (como se ha ido mostrando en el capítulo de *Marco teórico*), se construyen de forma similar con los nombres específicos.

Así, lo primero que hago es el algoritmo (en este caso el de Bayes ingenuo multinomial), y guardarlo en una variable. Luego, llamo a la función *fit*. Lo que hace esta función básicamente es coger las variables *X* e *y* de entrenamiento, y las entrena con el modelo o algoritmo seleccionado, siguiendo los pasos

específicos de cada uno, cuyo funcionamiento, como ya he mencionado, se especifica en el *Marco teórico*. He usado los parámetros para cada algoritmo por defecto, como puede ser, por ejemplo, para el caso de Bayes ingenuo multinomial: *alpha*, *fit\_priot*, o *class\_prior*. Estos distintos parámetros se explicarán mejor en el capítulo de *Análisis de resultados obtenidos*, donde se verán las razones del mejor o peor rendimiento de cada modelo dependiendo de los parámetros preestablecidos.

```
bayes_model = MultinomialNB()
bayes_model.fit(X_train, y_train)
```

*Tabla 29: Ejemplo de construcción del modelo de machine learning con los árboles de decisión, y su posterior entrenamiento*

Una vez entrenado el modelo, el siguiente paso lógico a realizar es probar si funciona bien con los datos de prueba. Para ello, llamo al método *predict* el cuál, usando los *inputs X* de los datos de prueba, calcula los *outputs* de salida, que son las etiquetas de *spam/ham* de los emails y de *phishing/no phishing* de las *urls*, y las guarda en la variable *y* nueva, como se observa en el código de la *Tabla 30*.

```
# predicción del valor objetivo a partir de las muestras del modelo
y_test_bayes = bayes_model.predict(X_test)
```

*Tabla 30: Código correspondiente a la predicción que hace el modelo de machine learning creado con nuevos datos (en este caso, los datos de prueba)*

Estas predicciones van a servir para observar si los *outputs* precedidos se corresponden con los *outputs y* verederos seleccionados como datos de prueba, y eso determinará cuál es la precisión de cada algoritmo en relación a los valores predichos con los valores reales. Esta precisión se va a ver y analizar en el siguiente apartado, donde se van a comparar los resultados de precisión de cada algoritmo y cuál ha resultado ser el mejor.

### 3.3.1.5 Comparación de algoritmos

Tras el entrenamiento intensivo de los cuatro modelos seleccionados, y de probarlos con los datos de prueba, he procedido a medir cuál de estos algoritmos ha sido el más eficiente, medido en el valor de precisión de cada algoritmo para los datos de entrenamiento y de pruebas, ordenados de mayor a menor, o dicho de otro modo, del algoritmo con mejor precisión, al algoritmo con peor precisión. El resultado al comparar todos los algoritmos ha sido el que se muestra en la *Tabla 31* para los datos de entrenamiento y de pruebas de los emails; y en la *Tabla 32* para los datos de entrenamiento y de pruebas de las *urls*. En el capítulo de *Análisis de resultados obtenidos* se analizará más en detalle los resultados, con las métricas adecuadas (como el *recall* o la *puntuación-f1*), y explicando las razones de porque el mejor algoritmo es el de Bayes Ingenuo.

	<b>Modelo ML</b>	<b>Precisión de entrenamiento</b>	<b>Precisión de pruebas</b>
1	Bayes Ingenuo	0.963264	0.9563984
2	Arboles de Decisión	0.991104	0.9378826
3	Bosques Aleatorios	0.91008	0.8977154
4	K-Vecinos Más Próximos	0.989568	0.7533224

Tabla 31: Tabla analítica de la precisión de los distintos algoritmos con los datos de entrenamiento y de prueba de los emails

	<b>Modelo ML</b>	<b>Precisión de entrenamiento</b>	<b>Precisión de pruebas</b>
1	Bayes Ingenuo	0.9781569	0.9675369
2	Árboles de Decisión	0.9999063	0.947156
3	K-Vecinos Más Próximos	0.9981839	0.9354101
4	Bosques Aleatorios	0.6401995	0.6396183

Tabla 32: Tabla analítica de la precisión de los distintos algoritmos con los datos de entrenamiento y de prueba de las urls

Tal como se puede observar, para ambos modelos, el mejor algoritmo haciendo comparativa y media de ambos datos de precisión, es el de Bayes Ingenuo, como bien se ha indicado arriba. Por lo tanto, lo siguiente que he procedido a realizar es guardar este algoritmo usando la librería *pickle*, en un archivo de formato *.dat*, indicando que se abra con el permiso de escritura *w*, y en modo binario *b* (que se corresponde al *wb* del *open*), tal como se muestra en el código de la Tabla 33.

```
# guardado del modelo de Bayes Ingenuo en el archivo correspondiente
pickle.dump(bayes_model, open("NaiveBayesClassifier_mails.pickle.dat", "wb"))
```

Tabla 33: Código usado para guardar el modelo ganador en un archivo, para poder ser usado por otras clases dentro de mismo proyecto

### 3.3.2 Análisis de correos de Gmail

Una vez he entrenado y probado los distintos algoritmos con los distintos datasets obtenidos, ahora es importante poder observar si dado un caso real, como el envío de un mensaje a través de un sistema de mensajería real como es *Gmail*, a la hora de leer un correo, se clasifica correctamente en un mensaje con contenido *spam*, *phishing*, o *ham*. Para realizar esto, he implementado una clase nueva en *Python JupyterLab* que he llamado *GmailReader.ipynb*. Esta clase tiene dos etapas principales: la lectura de los correos de *Gmail*, y una vez leídos y guardados en un documento *csv* (correspondiendo una fila por correo), voy a pasarlo por los modelos clasificadores, usando el algoritmo ganador (el de Bayes ingenuo) implementados para observar si se clasifican bien. Hay que tener en cuenta, que me he creado y enviado al *Gmail* un conjunto de correos que tienen las mismas dimensiones que el *dataset* que he usado de entrenamiento para cada una de las partes, tanto en *emails* como en *urls*, de forma que no se produzcan errores de dimensiones incompatibles, y se intenten buscar las mismas palabras dentro de las mismas dimensiones de *datasets*. En los subapartados siguientes se van a explicar cada uno de los procesos realizados para esto, con los trozos de código implementado más importantes para el entendimiento del funcionamiento de esta clase.

#### 3.3.2.1 Lectura de correos de la bandeja de entrada

Para leer los correos de *Gmail* de mi cuenta, lo primero que he hecho es crearme una cuenta secundaria de *Gmail* enfocada especialmente a este proyecto, ya que no me interesa que leer y almacenar correos de mi cuenta principal, puesto que contiene una cantidad de correos enorme, y esto podría llevar un tiempo de lectura, proceso, y almacenamiento bastante prolongado y pesado para la memoria de mi ordenador. Además, quiero mantener las cosas lo más simples posible para no perder tiempo incensario.

De esta forma, lo primero que he hecho es conectarme a esta nueva cuenta creada con la librería *imaplib* [60], y seleccionar los correos de la bandeja de entrada (*inbox*), así como seleccionar algunos elementos interesantes para la implementación del código, como pueda ser el último y primer correo de esta bandeja de entrada, o los *ids* de los emails. Esto se ve en el código de la *Tabla 34*.

Para conectarme a la cuenta, en este código de la *Tabla 34*, introduzco primeramente mi nombre de usuario de mi cuenta de *Gmail*. Luego, mi contraseña la leo automáticamente de un documento que he creado para almacenar exclusivamente la contraseña (antes de ponerla directamente en el código por temas obvios de privacidad y seguridad), llamado *contraseña.txt*. Finalmente, me conecto a la cuenta con el servidor y el puerto SMTP de la librería *imaplib* por defecto.

```
# -----
#
# Clase creada para leer emails del Gmail usando Python
#
# -----
#
# credenciales de cuenta
username = "a.vazquezgomez1996@gmail.com"
with open("contraseña.txt", "r") as fd:
    password = fd.read().strip()
SMTP_SERVER = "imap.gmail.com"
SMTP_PORT = 993
mail = imaplib.IMAP4_SSL(SMTP_SERVER)
mail.login(username,password)
mail.select('inbox')
data = mail.search(None, 'ALL')
mail_ids = data[1]
id_list = mail_ids[0].split()
first_email_id = int(id_list[0])
latest_email_id = int(id_list[-1])
url = ''
```

*Tabla 34: Código para conectarse a mi cuenta de Gmail*

Una vez conectada esta clase a esta cuenta de *Gmail*, el siguiente paso es leer los correos y extraerlos al documento *csv*. Para ello, lo primero que he hecho es con la función *fetch* de *email*, rescatar todos los mensajes recibidos en la cuenta de *Gmail*, desde el primero hasta el último de ellos, y guardarlos en una variable que he llamado *data*, en formato de email *RFC822* [61], que indica que cualquier palabra o carácter valido serán rescatados, siempre que formen parte del conjunto de caracteres *ASCII*.

Luego se comprueba que los emails rescatados son tuplas de objetos con la función *isinstance*, y en caso de ser así, se separan las distintas partes del mensaje en variables distintas, como puede ser el remitente (*email\_from*), el

asunto del mensaje (*email\_subject*), o el cuerpo del mensaje (*email\_body*). Para estos dos últimos (asunto y cuerpo del mensaje), he creado dos métodos auxiliares separados. El primero de ellos llamado *decode\_mime\_words*, transforma el texto del asunto del mensaje de *utf-8* a la codificación inglesa o castellana, ya que siendo en *utf-8*, lo más normal es que el asunto se muestre de la siguiente manera: =?utf-8?Q?hola?=; mientras que con este método quitaríamos la codificación *utf-8* alrededor del mensaje, quedando solo *hola*. El otro método auxiliar llamado *get\_body*, permite rescatar de forma limpia el cuerpo del mensaje, ya que no existe como tal ningún método de *Python* que proporcione el mensaje en la codificación que quiero. La proporciona en bruto en *utf-8* o en *latin-1*, con problemas a la hora de transcribirla al documento *csv*.

Una vez hecho esto, se quitan los acentos y símbolos especiales del asunto y cuerpo del mensaje con el método *unidecode*, ya que son molestos y no se transcriben bien al formato requerido por el documento *csv*. Luego, se ha creado la variable *urls*, que buscará *urls* en el cuerpo del mensaje con el método *search* de la clase *re*, con todos los elementos que contengan el elemento *regex* que incluya una *url* con *https*, acompañado de cualquier número de caracteres alfabéticos o numéricos. Se tiene que buscar primero en una variable para sacar estas *urls* en bruto, con el formato que da *re.search* por defecto, para poder sacar la *url* luego en limpio usando el mismo *seacrh* de *re*, con el método *group*, sin las *urls* nulas, ya que no *group* no las acepta y devolvería un error de método inexistente para valores nulos; de esta forma se obtienen las *url* tal como quería inicialmente: limpias y claras como las que hay en el *dataset* de entrenamiento usado en la clase de entrenamiento y pruebas de *urls*.

Finalmente, se transcribe todo al *csv*, en el que se han creado cuatro columnas: una para el remitente del mensaje, otra para el asunto, otra para el cuerpo del mensaje, y, por último, otra para las *urls* encontradas dentro del cuerpo del mensaje.

Todo este código se puede ver en la *Tabla 35*, y en la *Tabla 36* se puede observar un esbozo de la cabecera del documento *csv* creado, con las distintas columnas creadas de remitente (columna *from*), asunto del mensaje (columna *subject*), cuerpo del mensaje (columna *body*), y las *urls* encontradas en dicho cuerpo de mensaje (columna *urls*).

```
for i in range(latest_email_id,first_email_id-1, -1):
    data = mail.fetch(str(i), '(RFC822)' )
    for response_part in data:
        arr = response_part[0]
        if isinstance(arr, tuple):
            msg = email.message_from_string(str(arr[1]),'utf-8')
            email_subject_str = decode_mime_words(msg['subject'])
            email_subject Accent = str.join(" ", email_subject_str.splitlines())
            email_subject = unidecode.unidecode(email_subject Accent)
            email_from = msg['from']
            email_body_str = str(get_body(msg).decode('utf-8'))
```

```

email_body Accent = str.join(" ", email_body_str.splitlines())
email_body = unidecode.unidecode(email_body Accent)
urls_raw = re.search("(?P<url>https?://[^\\s]+)", email_body)
print(urls_raw)
if urls_raw is not None:
    urls = re.search("(?P<url>https?://[^\\s]+)", email_body).group("url")
    print(urls)
else:
    urls = ''
writer.writerow([email_from, email_subject, email_body, urls])
gmails_file.close()

```

Tabla 35: Código correspondiente a la lectura de Gmail y su almacenamiento en el documento csv creado

	<b>from</b>	<b>subject</b>	<b>body</b>	<b>urls</b>
0	alex_anyways <albalexbelen@gmail.com>	Fwd: Nota: Tu experiencia de 500 GB de espacio...	----- Forwarded message ----- De: Sun...	https://store.asuswebstorage.com/special-offer...
1	alex_anyways <albalexbelen@gmail.com>	Fwd: Kimberly_89 send a message	----- Forwarded message ----- De: Sam...	http://www.iptruster.com/lt/index.php/campaign...
2	Mike Ciolli <mikee@icis.com>	[*SPAM*] LOAN/FINANCING REQUEST OVERVIEW	<html> <head> \t<title></title> </head> <body>...	https://www.google.com/url?q=http://www.phxaff...
3	virusalert@efiltro.fi.upm.es	[*INFECTADO*] factura pendiente (correo de <f...	AVISO DE VIRUS: Se ha detectado, en un mensa...	http://code.google.com/a/apache-extras.org/p/p...
4	"Maria Martinez Gomez" <gildamlodzianowskim142...	***SPAM*** Espero que estes teniendo un excele...	En serio, no se que decir https://bit.ly/3pBFs...	https://bit.ly/3pBFs2i

Tabla 36: Esbozo de la cabecera documento csv creado con el remitente, asunto, cuerpo, y urls de los mensajes rescatados de mi cuenta de Gmail

En la siguiente subsección voy a explicar el último paso respecto a la implementación de código, que es la clasificación de estos emails rescatados, guardados en el documento csv.

### 3.3.2.2 Clasificación de los correos leídos

En esta última sección de desarrollo, voy a explicar el código correspondiente a como he catalogado los correos en aquellos que son de *phishing*, de *spam*, o de *ham*.

Lo primero que he realizado es cargar los modelos que había guardado en la sección de *Comparación de algoritmos* con la biblioteca *pickle* de *Python*, usando el método *pickle.load*. De esta forma he cargado y preparado el modelo de Bayes ingenuo ganador, para clasificar los emails de *Gmail* que he extraído en el csv.

Esta clasificación se consigue, igual que en el capítulo de *Entrenamiento de algoritmos*, con el método *predict*, utilizando las variables *input X* y *output y* que he conseguido tras la limpieza de los cuerpos de los correos y las *urls*, de la misma forma que como lo hice en la sección de *Entrenamiento de algoritmos*.

Para guardar estos resultados, he creado un *DataFrame* que he llamado *gmails\_predict*, que he igualado al *csv* del documento donde he guardado los *Gmails* con el método de *pandas pd.read\_csv*, y he creado dos nuevas columnas, para los resultados de las predicciones de *emails*, y de *urls*.

El código correspondiente se muestra en la *Tabla 37*.

```
# carga de los modelos a partir del archive creado con pickle en el código de la Tabla 33
loaded_model_mails = pickle.load(open("NaiveBayesClassifier_mails.pickle.dat", "rb"))

# load model from file

loaded_model_urls = pickle.load(open("NaiveBayesClassifier_urls.pickle.dat", "rb"))

url_predict = loaded_model_urls.predict(X_urls)

gmails_predict['url label'] = url_predict

email_predict = loaded_model_mails.predict(X_mails)

gmails_predict['mail label'] = email_predict
```

*Tabla 37: Código para cargar el modelo de Bayes Ingenuo y predecir los datos de los correos obtenidos de la cuenta de Gmail*

El último paso es mirar, para todas las filas del *DataFrame* creado, en las columnas de predicciones de *urls* y de *emails*, los valores resueltos. En caso de que haya una *url* en el cuerpo del mensaje, y esa *url* estuviese marcada con el valor 1, se clasifica directamente como *phishing*, se escribe el remitente, asunto, y cuerpo del mensaje en el archivo de texto creado *phishing\_gmails.txt* (que como el nombre indica, contiene solo los correos clasificados como *phishing*), y realizo un *break* para salirme del bucle, ya que no es necesario seguir catalogando el mensaje al ser clasificado de *phishing*.

En caso de que no exista una *url* o que esté esa *url* clasificado con un 0, se procede a analizar el texto en si del cuerpo para ver si es *spam* o *ham*. Si de la predicción del texto del email sale como resultado la etiqueta 1, el email se considera *spam*, y se escribe el remitente, asunto, y cuerpo del mensaje en el documento de texto creado para almacenar correos de tipo *spam*, *spam\_gmails.txt*.

En caso contrario a cualquiera de estos casos, el correo se considerará legítimo o *ham*, y se escribirá en el documento creado para albergar este tipo de correos llamado *ham\_gmails.txt*.

Luego de esto, simplemente se cerrarán todos los archivos, ya que se ha escrito todo lo que quería escribir en ellos. El código de lo anterior explicado se muestra en la *Tabla 38*. Además, se muestra en la *Figura 34* como es el email en su origen y un esbozo de cómo se escribe dicho *Gmail*, en la *Tabla 39*, en el documento correspondiente, que sería en este caso de *phishing*, al contener una *url* que lleva a una página peligrosa, y ha sido calificada como 1 en la etiqueta de *phishing/no phishing*.

```

for i, rows in gmails_predict.iterrows():

    if gmailto_predict['urls'][i] is not None:

        if gmailto_predict['url label'][i] is 1:

            phishing_gmails.write("De : " + gmailto_predict['from'][i]+ '\n')
            phishing_gmails.write("Asunto : " + gmailto_predict['subject'][i]+ '\n')
            phishing_gmails.write(gmailto_predict['content'][i]+ '\n')

            break

    if gmailto_predict['mail label'][i] is 1:

        spam_gmails.write("De : " + gmailto_predict['from'][i]+ '\n')
        spam_gmails.write("Asunto : " + gmailto_predict['subject'][i]+ '\n')
        spam_gmails.write(gmailto_predict['content'][i]+ '\n')

    else:

        ham_gmails.write("De : " + gmailto_predict['from'][i]+ '\n')
        ham_gmails.write("Asunto : " + gmailto_predict['subject'][i]+ '\n')
        ham_gmails.write(gmailto_predict['content'][i]+ '\n')

phishing_gmails.close()
spam_gmails.close()
ham_gmails.close()
gmailto_predict.close()

```

Tabla 38: Código creado para catalogar los emails en phishing, spam, o ham, dependiendo de los resultados obtenidos

\*\*\*SPAM\*\*\* Espero que estés teniendo un excelente día de trabajo Recibidos x



**Maria Martinez Gomez** gildamlodzianowskim142@outlook.com [a través de upm.es](#)  
para M, jramirez, sbernardos, iboguslavsky, iso, Vinciane, mariadelsocorro, jlmorant, Nik ▾

En serio, no sé qué decir <https://bit.ly/3pBFs2i>

Maria Martinez Gomez

Figura 34: Visualización del correo ejemplo puesto en la bandeja de entrada de Gmail cuando lo he abierto

From : "Maria Martinez Gomez" <gildamlodzianowskim142@outlook.com>

Subject : \*\*\*SPAM\*\*\* Espero que estes teniendo un excelente dia de trabajo

En serio, no se que decir <https://bit.ly/3pBFs2i> Maria Martinez Gomez

Tabla 39: Visualización en el documento de texto *phishing\_gmails.txt* del correo de la Figura 34 tras ser analizado por el modelo de Bayes ingenuo, y se clasificado como correo de tipo phishing

Tras probarlo con varios correos, he obtenido unos resultados relativamente buenos, con un 85% de aciertos, observando que emails eran catalogados como *phishing*, *spam*, y *ham*.

El siguiente capítulo se corresponderá al análisis de resultados, donde se analizarán las métricas de los distintos algoritmos, y por qué ha ganado el algoritmo de Bayes ingenuo; y también haré una breve conclusión al trabajo realizado, explicando algunos cambios que he tenido que ir realizando por el camino.

## 4 Resultados y conclusiones

En este apartado voy a hacer un resumen de resultados obtenidos en el TFG, y las conclusiones personales sobre el trabajo realizado.

### 4.1 Análisis de resultados obtenidos

En esta sección voy a analizar los resultados obtenidos de cada algoritmo. Para ello, voy a analizar las métricas obtenidas de la matriz de confusión, así como los valores de precisión, *recall*, puntuación-*f1*, o el soporte. Antes de empezar, voy a explicar que significan cada una de estas métricas y para qué sirven.

La **matriz de confusión** [62] se refiere a una técnica usada para resumir el rendimiento de un algoritmo de clasificación. Los aciertos y fallos en la clasificación de un modelo de *machine learning* pueden ser engañosos si se tiene un número desigual de observaciones por cada clase, lo que he intentado balancear, como ya dije en el capítulo de *Preprocesamiento de datos*, aunque pueden darse aún algunos errores. Si embargo tras probar el entrenamiento con los datos no balanceado, y con los datos balanceados, durante la etapa de desarrollo, he observado que el rendimiento es obviamente mejor cuando los datos estás balanceados.

De este modo, una matriz de confusión  $C$  es tal que  $C_{i,j}$  es igual al número de observaciones conocidas en el grupo  $i$  de los *datasets* que se tienen, y aquellas observaciones obtenidas de las predicciones del grupo  $j$ . De este modo, en la clasificación binaria, podemos contar estas observaciones en cuatro categorías: la categoría de negativos verdaderos o  $C_{0,0}$ ; la categoría de falsos negativos o  $C_{1,0}$ ; la categoría de positivos verdaderos o  $C_{1,1}$ ; y la categoría de falsos positivos o  $C_{0,1}$ . En el caso de los resultados analizados, esto se va a mostrar en una gráfica con cuatro cuadros, correspondientes a la relación entre los valores reales y predichos por cada uno de los modelos para sus respectivas clases.

La **precisión** mide prácticamente lo mismo que median las precisiones de la *Tabla 31* y *Tabla 32*. Esto es, la medida del número de aciertos que han conseguido las predicciones respecto a los valores reales. Explicado de una forma más meticulosa, es la ratio  $\frac{t_p}{(t_p + f_p)}$ , donde  $t_p$  es el número de positivos verdaderos, y  $f_p$  es el número de falsos positivos. La precisión es la habilidad del modelo clasificador de no etiquetar como positivo una muestra de los datos que es realmente negativa, siendo el mejor valor 1 y el peor 0.

El **recall** es la ratio  $\frac{t_p}{(t_p + f_n)}$ , siendo en este caso  $t_p$  el número de positivos verdaderos, y  $t_n$  el número de falsos negativos. Dicho de otro modo, el *recall* es la habilidad del clasificador de encontrar todas las muestras positivas dentro de los datos. Igual que con la precisión, el mejor valor 1 y el peor 0.

La **puntuación-f1**, también conocida como puntuación-*f* o medida-*f*, se puede interpretar como la media ponderada de la precisión y del *recall*, donde una puntuación-*f1* tiene su mejor valor en 1, y su peor en 0. La contribución relativa de la precisión y el *recall* a este parámetro es igual, y su fórmula es:

$$f1 = \frac{2 * (\text{precisión} * \text{recall})}{(\text{precisión} + \text{recall})}$$

El **soporte** es el número de casos que se dan para cada categoría dentro de la matriz de confusión, tanto casos positivos verdaderos, positivos falsos, negativos verdaderos, y negativos falsos.

Explicados estos distintos términos relativos a las métricas que voy a analizar, ahora voy a explicar los resultados para cada uno de los modelos en relación a las dos clases.

#### 4.1.1 Árboles de decisión

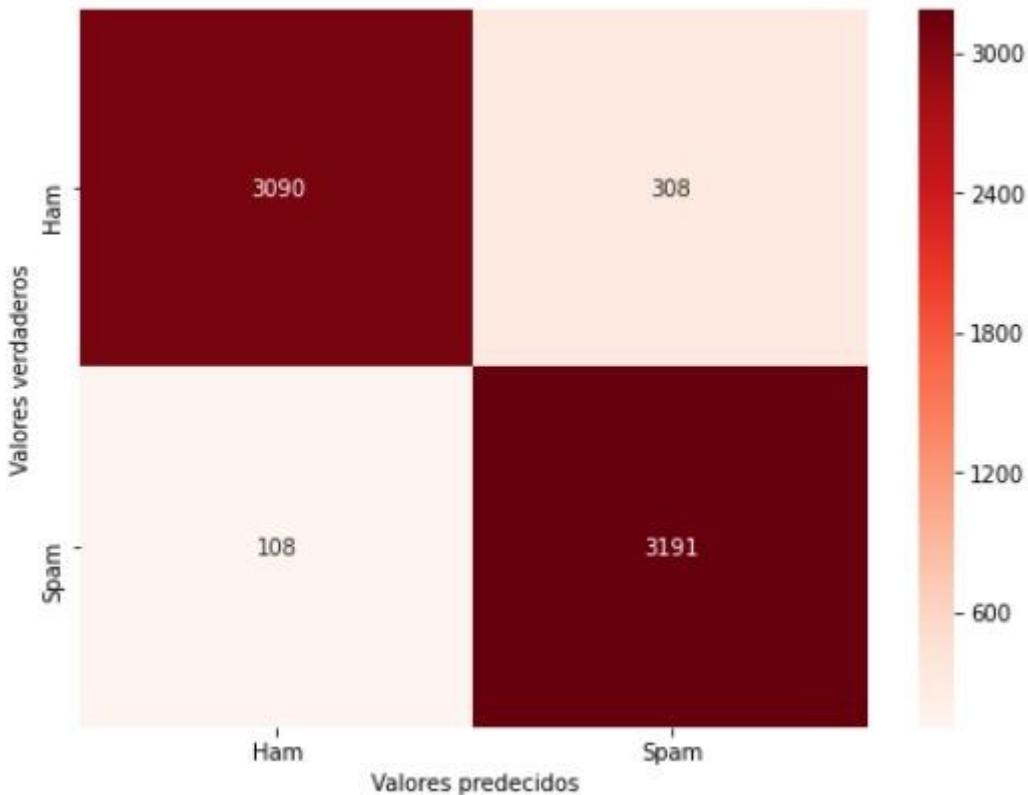


Figura 35: Matriz de confusión para las predicciones realizadas por el algoritmo de árboles de decisión en los datos de emails

En esta primera métrica de la Figura 35, se observa la matriz de confusión de los emails *ham/spam* habiendo usado el algoritmo de árboles de decisión. Como se puede observar, los árboles de decisión tienen un total de aciertos de 3090 para los correos de tipo *ham* (casos positivos verdaderos), y de 3191 para los correos de tipo *spam* (casos negativos verdaderos), haciendo esto un total de 6281 casos correctamente clasificados. Sin embargo, para los casos calificados

incorrectamente existen los correos *ham* clasificados como *spam* (casos positivos falsos) de un total de 308 calificados erróneamente, y el caso de correos *spam* clasificados como *ham* (casos negativos falsos), siendo estos un total de 108 correos etiquetados incorrectamente. Se puede ver entonces que los arboles de decisión tiene un rendimiento un poco mejor cuando se trata de los casos negativos de correos *spam*, que en los casos de correos positivos *ham*.

Vamos a ver ahora como se traduce esto en el informe de clasificación con las respectivas métricas, como se muestra en la *Tabla 40*.

	<b>precisión</b>	<b>recall</b>	<b>puntuación-f1</b>	<b>soporte</b>
0	0.91	0.97	0.94	3198
1	0.97	0.91	0.94	3499
aciertos			0.94	6697
media macro	0.94	0.94	0.94	6697
media ponderada	0.94	0.94	0.94	6697

*Tabla 40: Informe de clasificación del algoritmo de árboles de decisión en emails*

En esta *Tabla 40*, como se puede predecir, tenemos la fila 0 para las métricas de los correos *ham*, la fila 1 para las métricas de los correos *spam*, y el resto de filas para los cálculos, como puede ser la de ciertos para el número total de valores predichos correctamente, y la de media macro y media ponderada. La media macro [63] es la media de las puntuaciones *f1* de cada etiqueta creada, sin tener en consideración la proporción de cada etiqueta en el *dataset*. La media ponderada, por otro lado, es la media de esas puntuaciones *f1* por cada etiqueta, considerando la proporción de cada etiqueta.

Dicho esto, se observa entonces, como ya dije en la matriz de confusión que con los arboles de decisión, hay un mayor número de acierto o precisión en los correos *spam* que en los *ham*, aunque el *recall* sea mayor en estos debido a la entre positivos verdaderos y falsos negativos para cada clase. Aun así, la puntuación *f1* está muy igualada, con un 94% en total en aciertos totales.

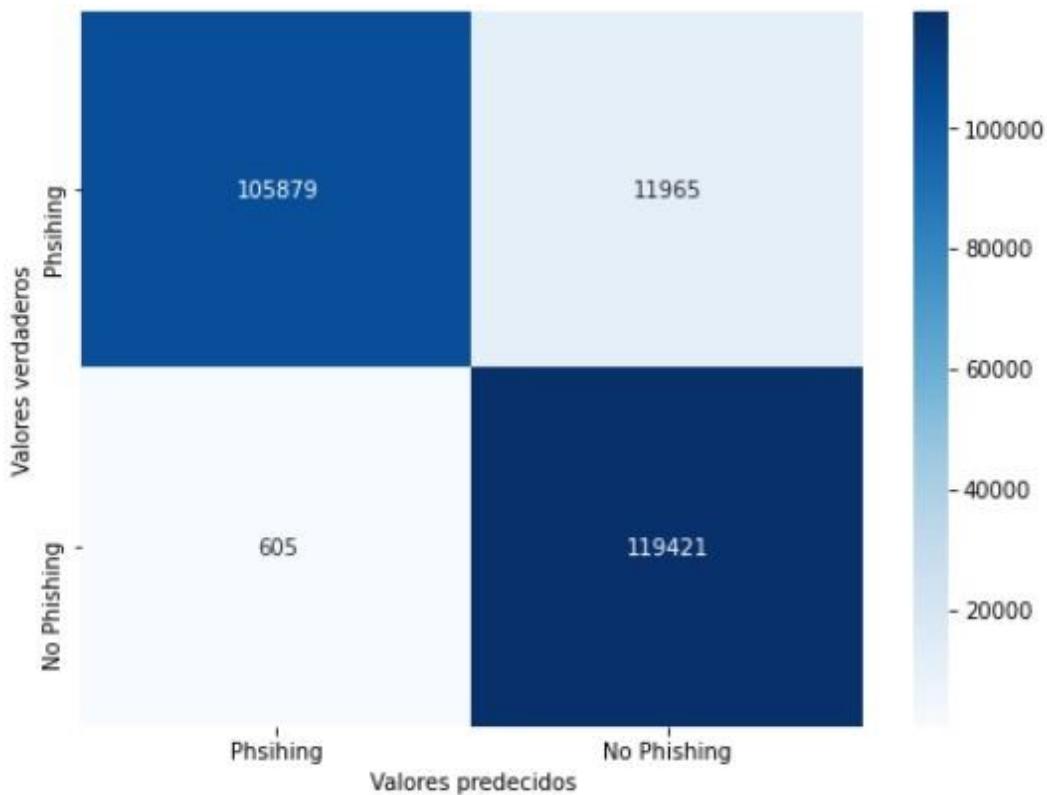


Figura 36: Matriz de confusión para las predicciones realizadas por el algoritmo de árboles de decisión en los datos de urls

En cuanto a la matriz de confusión de la Figura 36, que corresponde a la relación de *urls* predichas y reales *phishing* y no *phishing*, se puede observar unas mejores predicciones relativas en la *urls* de tipo no *phishing*, con un total de 119421 de *urls* de tipo no *phishing* clasificadas correctamente (casos positivos verdaderos), en relación a 605 de estas *urls* clasificadas como *phishing* (casos positivos falsos). Sin embargo, en las *urls* de tipo *phishing* se clasifican correctamente 105879 (casos negativos verdaderos), mientras que se clasifican erróneamente 11965 *urls* como no *phishing*, cuando son *phishing* (casos negativos falsos).

	<b>precisión</b>	<b>recall</b>	<b>puntuación-f1</b>	<b>sopporte</b>
1	0.90	0.99	0.94	106484
0	0.99	0.91	0.95	131386
aciertos			0.95	237870
media macro	0.95	0.95	0.95	237870
media ponderada	0.95	0.95	0.95	237870

Tabla 41: Informe de clasificación del algoritmo de árboles de decisión en urls

El informe de clasificación de la Tabla 41 indica unas mejores predicciones de los correos de tipo no *phishing*, aunque la relación entre los casos verdaderos y falsos del *recall* es mejor en las *urls* *phishing*. Igualmente, como se observa, el número de aciertos es bueno, incluso mejor que con los emails, de un 95%.

#### 4.1.2 K-vecinos más próximos

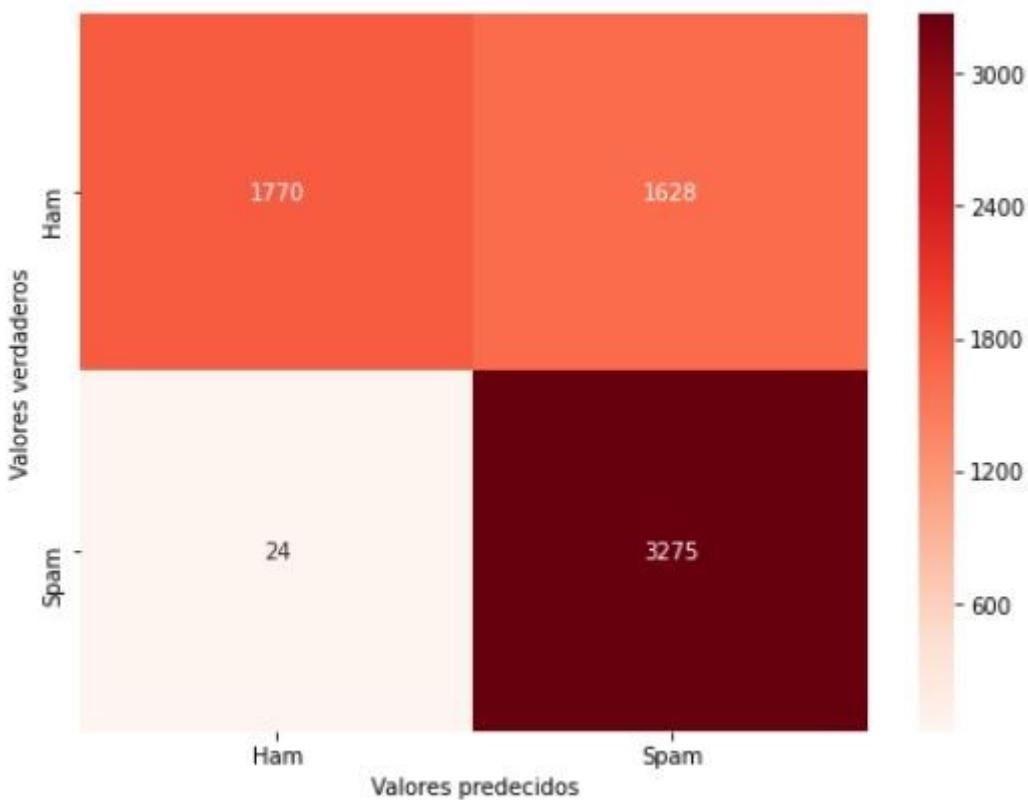


Figura 37: Matriz de confusión para las predicciones realizadas por el algoritmo de K-vecinos más próximos en los datos de emails

	<b>precisión</b>	<b>recall</b>	<b>puntuación-f1</b>	<b>soporte</b>
0	0.52	0.99	0.68	1794
1	0.99	0.67	0.80	4903
aciertos			0.75	6697
media macro	0.76	0.83	0.74	6697
media ponderada	0.87	0.75	0.77	6697

Tabla 42: Informe de clasificación del algoritmo de K-vecinos más próximos en emails

Como se puede observar en la Figura 37 y en la Tabla 42, el algoritmo de K-vecinos más próximos tiene un rendimiento mucho menor para la predicción de emails *ham/spam*, del 75% de aciertos, prediciendo mejor los emails *spam* que los *ham*. Teniendo en cuenta las diferentes categorías que ya se han visto en los resultados del algoritmo de árboles de decisión, se tienen un total de 1770 predicciones positivas verdaderas, 1628 predicciones positivas falsas, 24 predicciones negativas falsas, y 3275 predicciones negativas verdaderas. Hay una predicción con este algoritmo de K-vecinos más próximos bastante bajo en cuanto a correos *ham*, del 52%, por lo que, viendo estos resultados con los emails, está por clasificarse entre los peores modelos para clasificar emails. Veamos ahora como clasifica las *urls*.

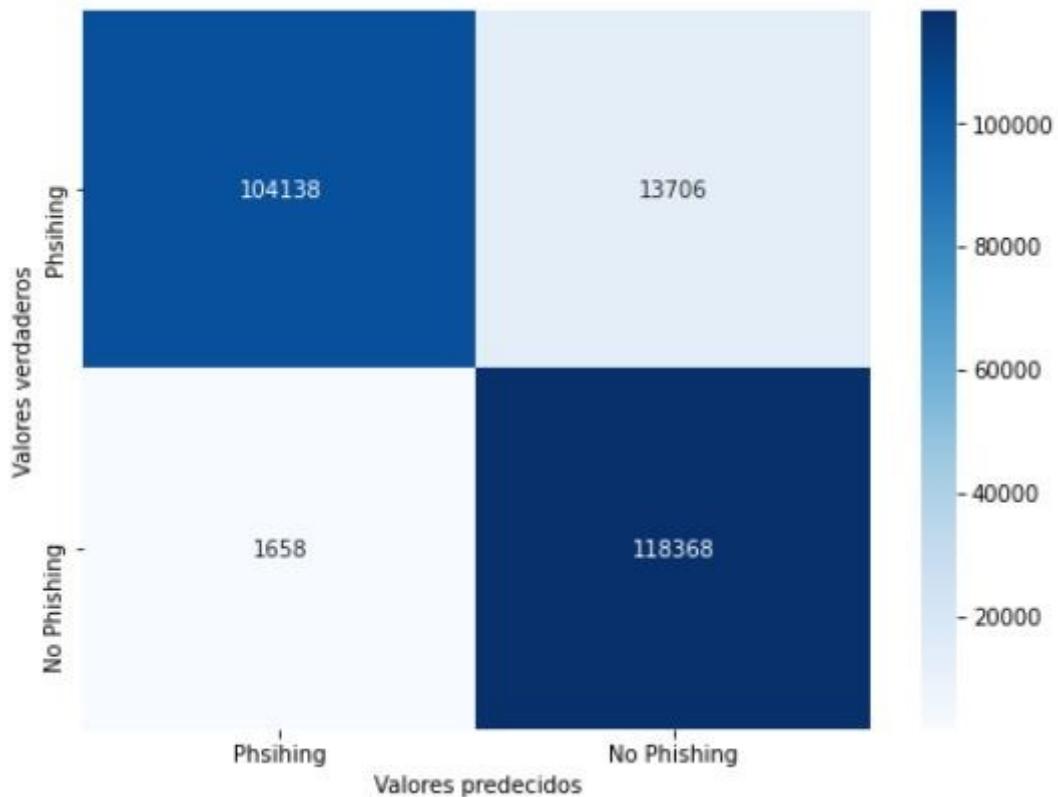


Figura 38: Matriz de confusión para las predicciones realizadas por el algoritmo de K-vecinos más próximos en los datos de urls

		precisión	recall	puntuación-f1	soporte
	1	0.88	0.98	0.93	105796
	0	0.99	0.90	0.94	132074
aciertos				0.94	237870
media macro		0.93	0.94	0.94	237870
media ponderada		0.94	0.94	0.94	237870

Tabla 43: Informe de clasificación del algoritmo de K-vecinos más próximos en urls

Como se observa en la Figura 38 y en la Tabla 43, el algoritmo de K-vecinos más próximos tienen un rendimiento mejor en la predicción de *urls* que en la de *emails*, por mucho, con un porcentaje de aciertos del 94%, aunque no supera al rendimiento de los árboles de decisión. Igual que los árboles de decisión, clasifica mejor en sus predicciones las *urls* de tipo no *phishing*, que las de tipo *phishing*, teniendo un total de 104138 casos positivos verdaderos, 13706 casos positivos falsos, 1658 casos negativos falsos, y 118368 casos negativos verdaderos.

#### 4.1.3 Bosques aleatorios

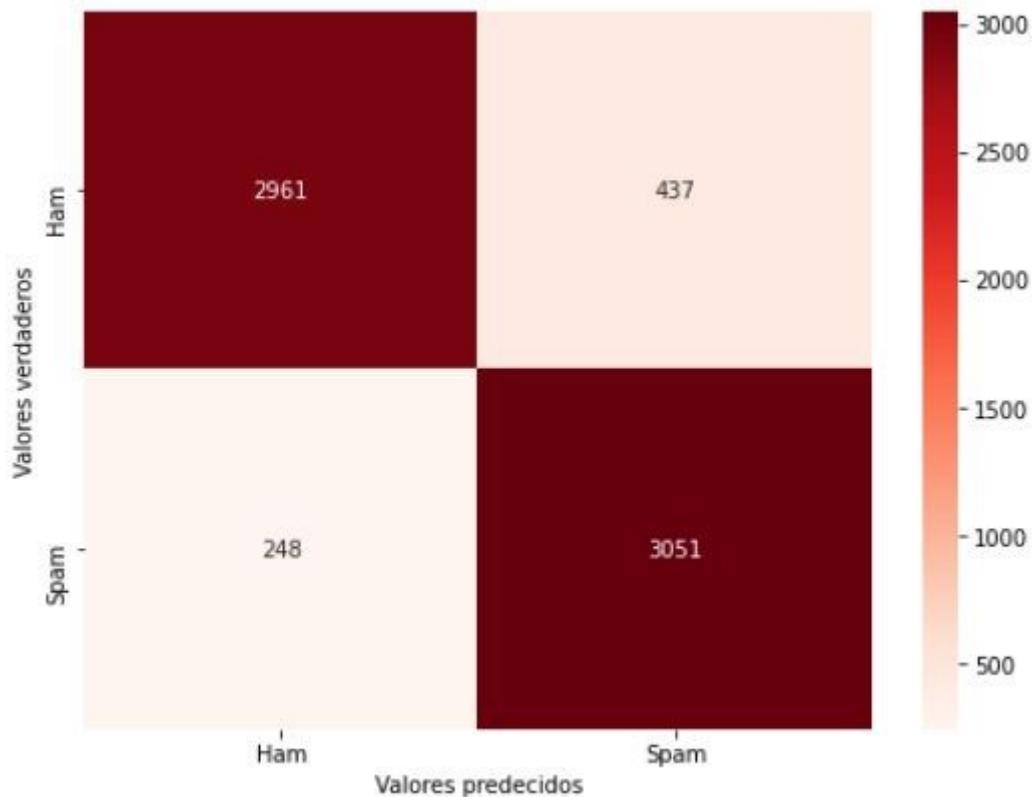


Figura 39: Matriz de confusión para las predicciones realizadas por el algoritmo de bosques aleatorios en los datos de emails

	precisión	recall	puntuación-f1	soporte
0	0.87	0.92	0.90	3209
1	0.92	0.87	0.90	3488
aciertos			0.90	6697
media macro	0.90	0.90	0.90	6697
media ponderada	0.90	0.90	0.90	6697

Tabla 44: Informe de clasificación del algoritmo de bosques aleatorios en emails

Como se observa en la Figura 39 y en la Tabla 44, los bosques aleatorios tienen un buen rendimiento con la clasificación de emails, con un 90% de aciertos, siendo por mucho mejor que el algoritmo de K-vecinos más próximos, pero un 4% peor que el de árboles de decisión. Clasifica mejor los correos *spam* que los *ham*, teniendo un total de 2961 casos positivos verdaderos, 437 casos positivos falsos, 248 casos negativos falsos, y 3051 casos negativos falsos. Este algoritmo de bosques aleatorios tiene pinta, de momento, de colocarse en una posición mejor que el de K-vecinos próximos, al menos en la clasificación de textos para determinar si el correo es *spam* o *ham*.

Pero vamos a ver ahora que tal es el rendimiento en la clasificación de *urls*.

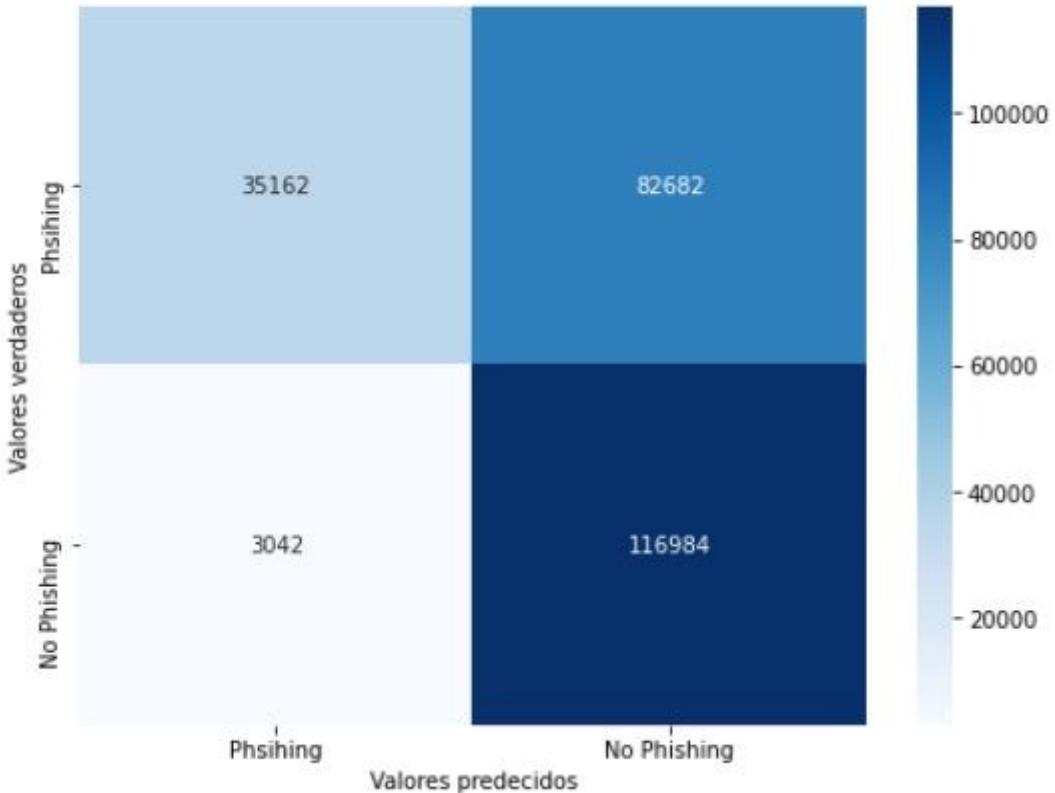


Figura 40: Matriz de confusión para las predicciones realizadas por el algoritmo de bosques aleatorios en los datos de urls

		<b>precisión</b>	<b>recall</b>	<b>puntuación-f1</b>	<b>soporte</b>
aciertos	1	0.30	0.92	0.45	38204
	0	0.97	0.59	0.73	199666
	media macro	0.64	0.75	0.59	237870
media ponderada		0.87	0.64	0.69	237870

Tabla 45: Informe de clasificación del algoritmo de bosques aleatorios en urls

Como se observa en la Figura 40 y en la Tabla 45, las predicciones de urls usando el algoritmos de bosques aleatorios son bastante nefastas, peor incluso que la de clasificación de emails en el algoritmo de K-vecinos más próximos, teniendo este caso un total en porcentaje, del 64% de aciertos totales para la clasificación de urls, y teniendo uno de los peores resultados de precisión, de un 30% para la clasificación de urls de tipo phishing, y como se observa, hay más casos positivos falsos que positivos verdaderos. En total, existen 35162 casos positivos verdaderos, 82682 casos positivos falsos, 3042 casos negativos falsos, y 116984 casos negativos verdaderos.

Ante todo esto me ha surgido una duda: ¿Cómo es posible que los bosques aleatorios tengan peores resultados que los arboles de decisión? Al fin y al cabo, los bosques aleatorios son algoritmos de ensamblado, que están compuestos por un conjunto determinado de árboles de decisión. Entonces, siendo este un algoritmo que tiene trabajando en conjunto varios árboles de decisión,

teóricamente parecería irrefutable que debería trabajar mejor que los arboles de decisión. Entonces, ¿Cómo es que no es así? La respuesta [64] a esta pregunta puede ser compleja y no concisa, dado que se pueden dar varios casos en los que un árbol de decisión puede dar mejores resultados que un bosque aleatorio. Por ejemplo, los arboles de decisión son más eficientes cuando se quiere crear un modelo simple, cuando el conjunto de *datasets* y características pueden ser usados por completo, cuando se tiene un poder computacional limitado, o cuando no hay mucha preocupación acerca de la precisión en *datasets* futuros. Esta última opción es la menos posible en el caso de este proyecto, pero si es cierto que los modelos creados no son excesivamente complejos, y el conjunto entero de *datasets* y características están disponibles para usar. Además, desde el ordenador portátil que he realizado el proyecto, si es cierto que hay una potencia de computación menor. Todas o algunas de estas razones pueden haber beneficiado a que los árboles de decisión tengan mejores resultados. Aún así, en general debería darse el caso normalmente de que los bosques aleatorios tengan una mejor precisión que los arboles de decisión, por lo que este caso que se ha dado en mi proyecto es bastante excepcional.

#### 4.1.4 Bayes Ingenuo

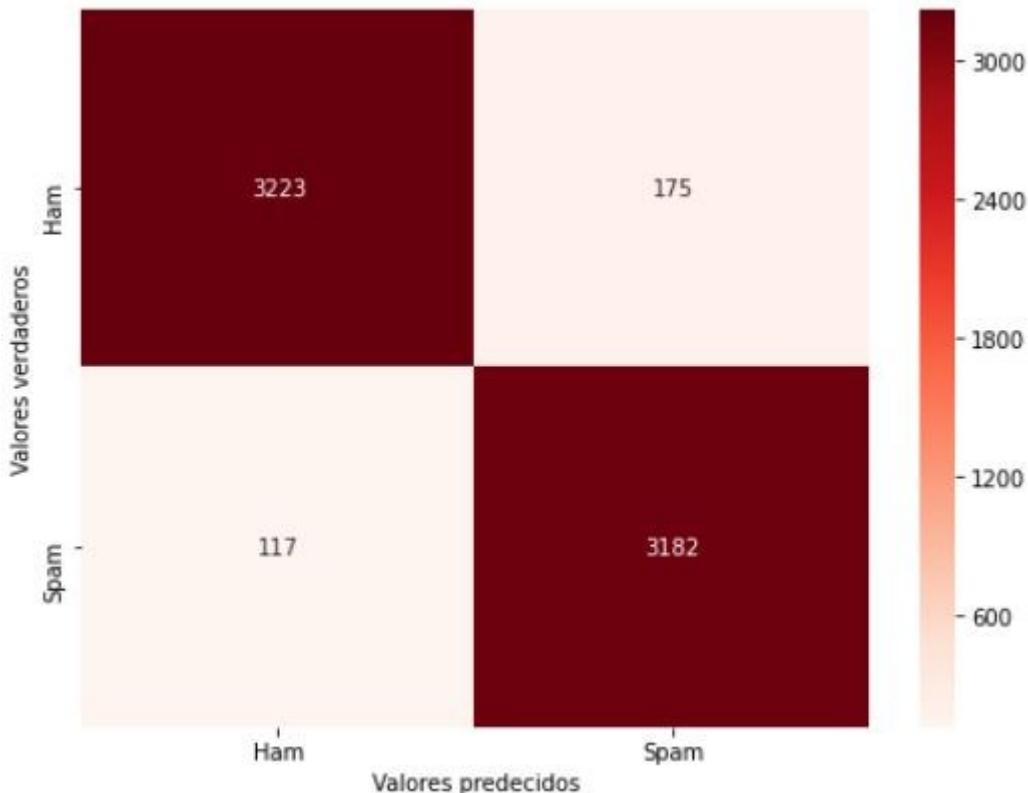


Figura 41: Matriz de confusión para las predicciones realizadas por el algoritmo de Bayes ingenuo en los datos de emails

	<b>precisión</b>	<b>recall</b>	<b>puntuación-f1</b>	<b>sopporte</b>
0	0.95	0.96	0.96	3340
1	0.96	0.95	0.96	3357
aciertos			0.96	6697

media macro	0.96	0.96	0.96	6697
media ponderada	0.96	0.96	0.96	6697

Tabla 46: Informe de clasificación del algoritmo de Bayes ingenuo en emails

Como se observa en la Figura 41 y en la Tabla 46, el modelo de Bayes ingenuo rinde bastante bien, con un total del 96% de aciertos, superando en este caso al 94% de aciertos que tenían los árboles de decisión. Además, tiene ratio de predicciones verdaderas. Tiene entonces para predicciones de emails *spam/ham* un total de 3223 casos positivos verdaderos, 175 casos positivos falsos, 117 casos negativos falsos, y 3182 casos negativos verdaderos.

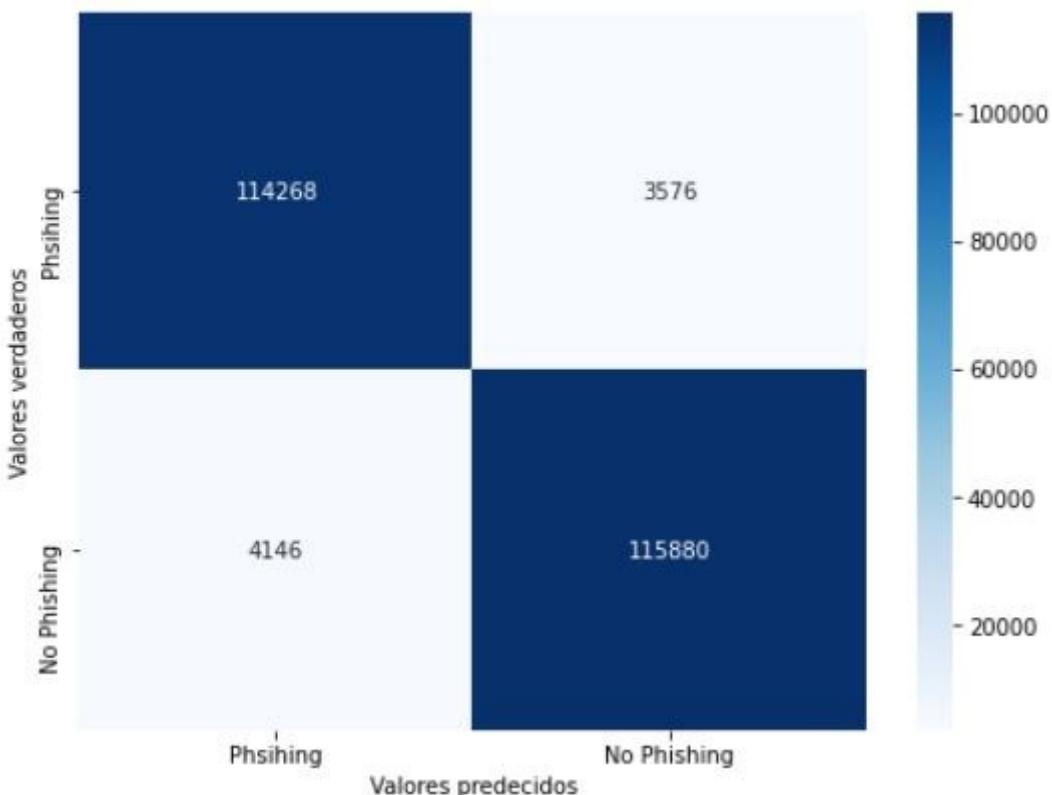


Figura 42: Matriz de confusión para las predicciones realizadas por el algoritmo de Bayes ingenuo en los datos de urls

	precisión	recall	puntuación-f1	soporte
1	0.97	0.96	0.97	118414
0	0.97	0.97	0.97	119456
aciertos			0.97	237870
media macro	0.97	0.97	0.97	237870
media ponderada	0.97	0.97	0.97	237870

Tabla 47: Informe de clasificación del algoritmo de Bayes ingenuo en urls

Como se observa en la Figura 42 y en la Tabla 47, también supera en las predicciones de *urls phishing* y *no phishing* a los árboles de decisión, con un 97% de aciertos frente al 95% que tenían los árboles de decisión, y teniendo un total de 114268 casos positivos verdaderos, 3576 casos positivos falsos, 4146

casos negativos falsos, y 115880 casos negativos verdaderos, demostrándose así que el modelo de Bayes ingenuo es el mejor modelo de los propuesto, pues es el que mejor precisión y ratio de aciertos tiene.

Las principales razones [65] de que haya ganado este algoritmo son que Bayes ingenuo tiene un muy buen rendimiento cuando trabaja con múltiples clases, y en clasificación de textos (como en el caso de este proyecto). Además, algunas de las ventajas de este modelo de Bayes ingenuo que han podido beneficiar a que tenga mejores rendimientos que el resto de modelos son:

- Es un algoritmo simple, y si el supuesto de independencia condicional se cumple, este modelo de Bayes ingenuo convergerá más rápido en sus clasificaciones que otros modelos, por lo que necesitará menos datos de entrenamiento para obtener mejores resultados. Este supuesto de independencia condicional [66] enuncia que, tras un condicionamiento en un conjunto de covariantes observadas, la asignación del tratamiento es independiente de los resultados potenciales. Esto se cumple incluso si la suposición de Bayes ingenuo no se cumple.
- El modelo de Bayes ingenuo requiere menos tiempo de entrenamiento del modelo para los *datasets* que se le proporcione.

Dicho esto, en la siguiente sección voy a realizar una breve conclusión del trabajo, haciendo inciso en lo que tenía planificado y que he tenido que cambiar a lo largo del proyecto para adaptarme a lo que he podido finalmente hacer.

## 4.2 Conclusiones

Como primera conclusión del trabajo realizado, cabe señalar la construcción del sistema que se planteaba. No obstante, merece la pena mencionar que no se han podido completar todas las tareas como se había planeado al principio (véase Anexo I). Por ejemplo, la revisión de la bibliografía ha durado más tiempo del pensado inicialmente puesto que había mucha información que analizar para el proyecto y mis conocimientos previos de *machine learning* eran bastante limitados. He tenido que investigar bastante al respecto para entender mejor las bases teóricas de esta y cómo funciona en la práctica.

Por otro lado, la elaboración del corpus no me ha llevado tanto tiempo como el que pensaba puesto que pude encontrar *datasets* ya fabricados en Internet, y usarlos como datos de entrenamiento y de prueba. No ha sido necesario que fabricase los datos de los correos por mi cuenta, teniendo acceso a ellos en Internet. Sin embargo, aquí he tenido un gran contratiempo, y es que al principio este proyecto lo había planteado para identificar solo correos de tipo *phishing*, pero al no haber podido encontrar un *dataset* completo de corpus específicos de correos de tipo *phishing*, he tenido que usar, como bien planteé en el capítulo de desarrollo, varios *dataset* que pertenecen a *emails spam*, *emails ham*, y *urls* de tipo *phishing* y de tipo legítimo. He tenido que crear por lo tanto dos modelos, uno entrenado con correos para clasificar los cuerpos del correo en correos tipo *spam* o *ham*, y otro entrenado con *urls* para observar si el correo es *phishing* o no, esto antes de entrar a clasificarlos en *spam* o *ham*.

Finalmente, la implementación del código se ha retrasado y acortado un poco debido a que, como he dicho anteriormente, el tema de documentación ha sido más denso y largo, y el código, gracias a las librerías que estoy usando de *scikit-learn*, es bastante sencillo de hacer, puesto que esta librería tiene ya los algoritmos prácticamente hechos, y solo hay que preocuparse de los pasos previos, como es la carga o preprocesamiento de los datos, y hacer las pruebas para estos datos.

El resto de las tareas se han mantenido prácticamente igual a como estaban planificadas.

Por último, me gustaría señalar algunos aspectos que han quedado fuera del trabajo realizado debido a las restricciones de tiempo y que servirían para poder tener más modelos entre los que comparar y poder estar más seguros de elegir el mejor. Uno de ellos, y posiblemente uno de los más importantes, ha sido no utilizar métodos de hiper-parámetros y *cross-validation*, para poder sacar un mejor rendimiento de los distintos algoritmos, y obtener resultados aún mejores de los que he obtenido aquí. En un futuro, si sigo trabajando en este proyecto, me gustaría implementar esta etapa en el código y mejorar mis resultados. Otro aspecto dejado fuera también ha sido crear modelos con otros algoritmos más potentes de aprendizaje automático, como redes neuronales, *SVM* (*Support Vector Machines*), o regresión lineal. Me gustaría trabajar también en un futuro con estos algoritmos, y compararlos con los ya realizados. También se podrían crear nuevos modelos cambiando la forma en que se manejan los textos, considerando, por ejemplo, no solo la frecuencia de las palabras, sino también información morfosintáctica.

Dicho esto, voy a realizar ahora un análisis de impacto de mi proyecto.

## 5 Análisis de Impacto

En este apartado se va a realizar un análisis del impacto potencial de los resultados obtenidos durante la realización de este trabajo, en los diferentes contextos para los que se aplique, que se muestran en los siguientes subapartados. He tenido en cuenta, dentro de lo que puede aportar mi proyecto, los distintos objetivos de desarrollo sostenible establecidos por *Naciones Unidas* [67]

### 5.1 Impacto personal

En lo referente a impacto personal, el trabajo realizado en mi TFG es útil en cuanto a la protección de datos personales frente a posibles ataques de tipo malicioso, como en el caso de nuestro proyecto, de tipo *phishing*. Cada persona individualmente puede no ser capaz de distinguir si un correo puede suponer una amenaza, y es cuando entran en acción los modelos creados en el trabajo para poder ayudar a determinar si son de tipo *phising* antes de que la víctima potencial los abra, pinche el enlace, y se le robe información delicada y confidencial de forma indirecta, como puedan ser contraseñas importantes, el número de su cuenta de crédito, o información delicada de carácter personal con la que el atacante pueda chantajear al usuario víctima.

En este ámbito, el trabajo realizado puede relacionarse de alguna forma con los objetivos 3, relacionado con garantizar una vida sana y promover el bienestar para todos en todas las edades [68]; y con el objetivo 4, relacionado con la educación de calidad [69]. En el objetivo 3 he encontrado cierta utilidad en mi trabajo al cumplimiento del punto 3.b, el cual enuncia: *Reforzar la capacidad de todos los países, en particular los países en desarrollo, en materia de alerta temprana, reducción de riesgos y gestión de los riesgos para la salud nacional y mundial*, puesto que el sistema creado de aprendizaje automático puede generar un mejor bienestar personal al no tener el riesgo de una mayor preocupación por posibles chantajes o robo de identidad.

En cuanto al objetivo 4, he encontrado que mi trabajo puede ser aplicable a la meta 4.a que enuncia: *Construir y adecuar instalaciones educativas que tengan*

*en cuenta las necesidades de los niños y las personas con discapacidad y las diferencias de género, y que ofrezcan entornos de aprendizaje seguros, no violentos, inclusivos y eficaces para todos*, puesto que el trabajo realizado puede aplicarse a evitar riesgos educativos de alumnos que entrasen por desconocimiento en correos o enlaces *phishing*, si se aplicase por ejemplo el sistema de *machine learning* creado a cuentas de correo creadas para una determinada escuela; y a la meta 4.c, que enuncia: *De aquí a 2030, aumentar considerablemente la oferta de docentes calificados, incluso mediante la cooperación internacional para la formación de docentes en los países en desarrollo, especialmente los países menos adelantados y los pequeños Estados insulares en desarrollo*, puesto que el trabajo realizado en mi TFG podría servir para enseñar a los profesores una nueva herramienta que evite los riesgos en enlaces o correos que tengan peligro de albergar contenido malicioso de tipo *phishing*.

## 5.2 Impacto empresarial

En cuanto al impacto empresarial, es importante destacar que el uso de este proyecto en el ámbito laboral puede ser de una gran utilidad, puesto que evitar que a los empleados de una empresa le lleguen correos con contenido malicioso de tipo *phishing* puede ahorrar que la empresa en su conjunto sea atacada de forma premeditada. Esto contribuye de forma indirecta a un mayor crecimiento económico, pues las empresas que reciben este tipo de ataques pueden verse obligadas a aceptar sobornos y perder grande sumas de dinero, lo cual el sistema de detección de mensaje de tipo *phishing* creado en este proyecto pretende evitar.

De esta forma, en este marco, el trabajo realizado en mi TFG puede relacionarse con el objetivo 8, relacionado con promover el crecimiento inclusivo y sostenible, el empleo y el trabajo decente para todos [70], puesto que un trabajo como el realizado puede ayudar bastante a crear un entorno de trabajo más seguro para todos. Se puede relacionar en especial con la meta 8.8 de este objetivo, que enuncia: *Proteger los derechos laborales y promover un entorno de trabajo seguro y sin riesgos para todos los trabajadores, incluidos los trabajadores migrantes, en particular las mujeres migrantes y las personas con empleos precarios*, dado que el sistema realizado en el trabajo reduce riesgos, y cuanto menor sea el riesgo laboral para los trabajadores y más se haga por protegerlos y apoyarlos, más se apoyarán sus derechos laborales y se promoverá un entorno de trabajo seguro. Esta es, seguramente, la medida que más he tenido en cuenta a la hora de realizar mi TFG, pues mi planteamiento siempre ha sido enfocarlo hacia un ámbito empresarial en el que se puedan proteger en el trabajo a las personas para evitar que corran riesgos innecesarios.

## 5.3 Impacto social

En cuanto al impacto social, el trabajo realizado en mi TFG puede ser bastante beneficioso, puesto que, igual que aporta bienestar a cada persona individualmente y le proporciona seguridad, también puede proporcionar bienestar social ante posibles ataques de tipo *phishing* graves que afecten de alguna forma a una sociedad completa.

De este modo, he podido encontrar cierta relación entre mi proyecto con el objetivo 11, relacionado con lograr que las ciudades sean más inclusivas, seguras, resilientes y sostenibles [71].

He podido relacionar especialmente mi proyecto con la meta 11.5, que enuncia: *De aquí a 2030, reducir significativamente el número de muertes causadas por los desastres, incluidos los relacionados con el agua, y de personas afectadas por ellos, y reducir considerablemente las pérdidas económicas directas provocadas por los desastres en comparación con el producto interno bruto mundial, haciendo especial hincapié en la protección de los pobres y las personas en situaciones de vulnerabilidad*, puesto que el trabajo de este TFG pretende proteger socialmente a los más vulnerables frente a desastres ocasionados por ataques de tipo *phishing*. De esta manera, siguiendo el mismo razonamiento, podría relacionarse con la meta 11.b, que enuncia: *De aquí a 2020, aumentar considerablemente el número de ciudades y asentamientos humanos que adoptan e implementan políticas y planes integrados para promover la inclusión, el uso eficiente de los recursos, la mitigación del cambio climático y la adaptación a él y la resiliencia ante los desastres, y desarrollar y poner en práctica, en consonancia con el Marco de Sendai para la Reducción del Riesgo de Desastres 2015-2030, la gestión integral de los riesgos de desastre a todos los niveles*.

## 5.4 Impacto económico

En el ámbito económico, se podrían ligar de cierta forma los objetivos de impacto empresarial, puesto que tienen conexión, dado que, por ejemplo, cuanto más dinero y trabajo puedan ofrecer dichas empresas, mejor mercado laboral habrá, y esto afectará directamente a una mejor economía mundial y territorial dentro de cada país.

Otros objetivos que se pueden relacionar con el impacto económico son el 9, relacionado con construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación [72]; y el 10, relacionado con reducir la desigualdad en y entre los países [73].

Con respecto al objetivo 9, he encontrado varias metas que se podrían relacionar con el trabajo del TFG, pero las que principalmente puedo relacionar son las metas 9.4, que enuncia: *De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas*, dado que especifica la adaptación de nuevas tecnologías que puedan ser más eficaces, y que sean limpias, metas que cumple el proyecto de este TFG entre manos; y la meta 9.c, que enuncia: *Aumentar significativamente el acceso a la tecnología de la información y las comunicaciones y esforzarse por proporcionar acceso universal y asequible a Internet en los países menos adelantados de aquí a 2020*, puesto que al dar acceso a más personas a nuevas tecnologías, gente que pueda ser más vulnerable por desconocimiento técnica y específico de ellas, confronte riesgos al utilizar dichas tecnologías, que el trabajo de este TFG precisamente he diseñado para evitar dichos riesgos.

En relación al objetivo 10, encuentro ciertas similitudes especialmente con la meta 10.5, que enuncia: *Mejorar la reglamentación y vigilancia de las instituciones y los mercados financieros mundiales y fortalecer la aplicación de esos reglamentos*, dado que una mayor vigilancia en posibles fraudes originados por ataques en instituciones importantes pueden ser importantes, dados a evitar que estos se produzcan en un principio creando un sistema que de seguridad frente a posibles ataques institucionales como pueden darse con los ataques de tipo *phishing* que desarrolla el trabajo de este TFG.

## **5.5 Impacto medioambiental**

En cuanto a términos medioambientales, no he encontrado ningún objetivo al que mi trabajo pueda aportar de ninguna forma, y no he enfocado ni veo ningún término de mi trabajo a este impacto, aunque si hubiese encontrado algún uso al respecto, lo habría aplicado sin dudarlo.

## **5.6 Impacto cultural**

En cuanto al impacto cultural, el trabajo realizado en este TFG también es importante, dado que promueve que no se promueva la violencia de ninguna forma contra etnias distintas amenazándolos o chantajeándolos con estos posibles ataques de tipo *phishing*.

De esta forma, relaciono el impacto cultural con el objetivo 16, relacionado con promover sociedades justas, pacíficas e inclusivas [74].

Este es quizás el objetivo con las metas, aparte de las empresariales, para las que el proyecto de este TFG estaban más enfocadas a cubrir, pues cumple varias metas de este objetivo, relacionadas con evitar violencia, fraude, terrorismo, y delincuencia, relacionado con metas como puedan ser la 16.4 en relación a la lucha de crimen organizado, que enuncia: *De aquí a 2030, reducir significativamente las corrientes financieras y de armas ilícitas, fortalecer la recuperación y devolución de los activos robados y luchar contra todas las formas de delincuencia organizada*; como puede ser también la 16.5, relacionada con la prevención de todo tipo de sobornos, que enuncia: *Reducir considerablemente la corrupción y el soborno en todas sus formas*; o la meta 16.a, la cual enuncia: *Fortalecer las instituciones nacionales pertinentes, incluso mediante la cooperación internacional, para crear a todos los niveles, particularmente en los países en desarrollo, la capacidad de prevenir la violencia y combatir el terrorismo y la delincuencia*. También puede haber cierta relación indirecta con la meta 16.10, que enuncia: *Garantizar el acceso público a la información y proteger las libertades fundamentales, de conformidad con las leyes nacionales y los acuerdos internacionales*, puesto que el dar acceso público a cierta información puede llegar a ser peligroso si no se protege de alguna forma, frente a ataques de carácter malicioso por la red, que cumple de esta forma con la protección de la libertades fundamentales de conformidad, como bien dice la meta.

## 6 Bibliografía

- [1] M. Ltd., «¿Que es el phishig? Tipos de phishing y ejemplos,» [En línea]. Available: <https://es.malwarebytes.com/phishing/>. [Último acceso: Marzo 2021].
- [2] I. Belcic, «Qué es el spam: guía esencial para detectar y prevenir el spam,» 8 Enero 2020. [En línea]. Available: <https://www.avast.com/es-es/c-spam#topic-1>. [Último acceso: Marzo 2021].
- [3] C. Systems, «What Is Phishing? Examples and Phishing Quiz,» [En línea]. Available: <https://www.cisco.com/c/en/us/products/security/email-security/what-is-phishing.html>. [Último acceso: Marzo 2021].
- [4] P. Protection, «History of Phishing: How Phishing Attacks Evolved From Poorly Constructed Attempts To Highly Sophisticated Attacks,» [En línea]. Available: <https://www.phishprotection.com/resources/history-of-phishing/>. [Último acceso: Marzo 2021].
- [5] Esquire, «SECRETS OF THE LITTLE BLUE BOX,» [En línea]. Available: <https://classic.esquire.com/article/1971/10/1/secrets-of-the-blue-box>. [Último acceso: 26 Abril 2021].
- [6] i. governance, «The 5 most common types of phishing attack,» [En línea]. Available: <https://www.itgovernance.eu/blog/en/the-5-most-common-types-of-phishing-attack>. [Último acceso: Abril 2021].
- [7] C. Security, «DNC Hacks: How Spear Phishing Emails Were Used,» [En línea]. Available: <https://www.calyptix.com/top-threats/dnc-hacks-how-spear-phishing-emails-were-used/>. [Último acceso: Abril 2021].
- [8] Wikipedia, «Comité Nacional Demócrata,» [En línea]. Available: [https://es.wikipedia.org/wiki/Comit%C3%A9\\_Nacional\\_Dem%C3%ADcrata](https://es.wikipedia.org/wiki/Comit%C3%A9_Nacional_Dem%C3%ADcrata). [Último acceso: Marzo 2021].
- [9] A. INTERNET, «Backlink,» [En línea]. Available: <https://www.atinternet.com/es/glosario/backlink/>. [Último acceso: Marzo 2021].
- [10] Wikipedia, «Tecnología push,» [En línea]. Available: [https://es.wikipedia.org/wiki/Tecnolog%C3%ADa\\_push](https://es.wikipedia.org/wiki/Tecnolog%C3%ADa_push). [Último acceso: Marzo 2021].
- [11] F. Dreamer, «Ham v Spam: what's the difference?,» 3 Octubre 2013. [En línea]. Available: <https://blog.barracuda.com/2013/10/03/ham-v-spam-whats-the-difference/>. [Último acceso: Marzo 2021].
- [12] Wikopedia, «SpamBayes,» [En línea]. Available: <https://en.wikipedia.org/wiki/SpamBayes>. [Último acceso: Marzo 2021].
- [13] Wikipedia, «Request for Comments,» [En línea]. Available: [https://es.wikipedia.org/wiki/Request\\_for\\_Comments](https://es.wikipedia.org/wiki/Request_for_Comments). [Último acceso: Marzo 2021].
- [14] P. Flach, «The ingredients of machine learning,» de *Machine Learning: The art and science of algorithms that make sense of data*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 13-48.

- [15] BBVA, «'Machine learning': ¿qué es y cómo funciona?», [En línea]. Available: <https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/>. [Último acceso: Abril 2021].
- [16] A. P. Engelbrecht, Computational Intelligence: An Introduction, Univeristy of Pretoria, South Africa: Wiley Online Library, 2007.
- [17] diegocampoh, «diegocampoh/MachineLearningPhishing», [En línea]. Available: <https://github.com/diegocampoh/MachineLearningPhishing>. [Último acceso: Marzo 2021].
- [18] Xoriant, «Decision Trees for Classification: A Machine Learning Algorithm,» [En línea]. Available: <https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html>. [Último acceso: Abril 2021].
- [19] Merkle, «El algoritmo K-NN y su importancia en el modelado de datos,» [En línea]. Available: <https://www.merkleinc.com/es/es/blog/algoritmo-knn-modelado-datos>. [Último acceso: Abril 2021].
- [20] GeeksforGeeks, «Naive Bayes Classifiers,» [En línea]. Available: <https://www.geeksforgeeks.org/naive-bayes-classifiers/>. [Último acceso: Abril 2021].
- [21] T. D. Science, «Understanding Random Forest,» [En línea]. Available: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>. [Último acceso: Abril 2021].
- [22] P. Flach, «Concept Learning,» de *Machine Learning: The art and science of algorithms that make sense*, Cambrisge University Press, Cambridge, Cambridge, 2012, pp. 104-128.
- [23] P. Flach, «Tree Models,» de *Machine Learning: The art and science of algorithms that make sense*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 129-156.
- [24] P. Flach, «Decision Trees,» de *Machine Learning: The art and science of algorithms that make sense*, Cabridge University Press, Cambridge, Cambridge, 2012, pp. 133-138.
- [25] P. Flach, «Distance-based models,» de *Machine Learning: The art and science of algorithms that make sense*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 231-261.
- [26] P. Flach, «Nearest-neighbour classification,» de *Machine Learning: The art and science of algorithms that make sense*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 242-245.
- [27] P. Flach, «Probabilistic models,» de *Nearest-neighbour classification*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 262-297.
- [28] P. Flach, «Probabilistic models for categorical data,» de *Machine Learning: The art and science of algorithms that make sense of data*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 273-282.
- [29] HTAGlossary.net, «curva (ROC) de la característica operativa del receptor (n.f.),» [En línea]. Available: [http://htaglossary.net/curva-\(ROC\)-de-la-caracter%C3%ADstica-operativa-del-receptor-\(n.f.\)](http://htaglossary.net/curva-(ROC)-de-la-caracter%C3%ADstica-operativa-del-receptor-(n.f.)). [Último acceso: Marzo 2021].

- [30] P. Flach, «Model ensembles,» de *Machine Learning: The art and science of algorithms that make sense of data*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 330-342.
- [31] P. Flach, «Bagging and random forests,» de *Machine Learning: The art and science of algorithms that make sense of data*, Cambridge University Press, Cambridge, Cambridge, 2012, pp. 331-333.
- [32] Wikipedia, «Agregación de bootstrap,» [En línea]. Available: [https://es.wikipedia.org/wiki/Agregaci%C3%B3n\\_de\\_bootstrap](https://es.wikipedia.org/wiki/Agregaci%C3%B3n_de_bootstrap). [Último acceso: Abril 2021].
- [33] Nitisha, «Email Spam Dataset,» 08 Diciembre 2020. [En línea]. Available: <https://www.kaggle.com/nitishabharathi/email-spam-dataset>. [Último acceso: Abril 2021].
- [34] T. Tiwari, «Phishing Site URLs,» 03 Agosto 2020. [En línea]. Available: <https://www.kaggle.com/taruntiwarihp/phishing-site-urls>. [Último acceso: Abril 2021].
- [35] JupyterLab, «Overview,» [En línea]. Available: [https://jupyterlab.readthedocs.io/en/stable/getting\\_started/overview.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html). [Último acceso: Abril 2021].
- [36] aprendeIA, «Introducción a la librería NumPy de Python,» [En línea]. Available: <https://aprendeia.com/introduccion-a-numpy-python-1/>. [Último acceso: Marzo 2021].
- [37] A. c. Alf, «La librería Pandas,» [En línea]. Available: <https://aprendeonalf.es/docencia/python/manual/pandas/>. [Último acceso: Marzo 2021].
- [38] J. Hunter, D. Dale, E. Firing y M. Droettboom, «Pyplot tutorial,» 8 Abril 2020. [En línea]. Available: <https://matplotlib.org/stable/tutorials/introductory/pyplot.html>. [Último acceso: Mayo 2021].
- [39] M. Ebrahim, «Tutorial de NLP con Python NLTK (ejemplos simples),» 21 Septiembre 2017. [En línea]. Available: <https://likegeeks.com/es/tutorial-de-nlp-con-python-nltk/>. [Último acceso: Abril 2021].
- [40] K. Katari, «Seaborn: Python,» 11 Agosto 2020. [En línea]. Available: <https://towardsdatascience.com/seaborn-python-8563c3d0ad41>. [Último acceso: Abril 2021].
- [41] Y. Holtz, «Wordcloud,» [En línea]. Available: <https://www.python-graph-gallery.com/wordcloud/>. [Último acceso: Abril 2021].
- [42] Codecademy, «What is Scikit-Learn?,» [En línea]. Available: <https://www.codecademy.com/articles/scikit-learn>. [Último acceso: Marzo 2021].
- [43] T. P. S. Foundation, «19.1. email — An email and MIME handling package,» 18 Diciembre 2020. [En línea]. Available: <https://docs.python.org/3.5/library/email.html>. [Último acceso: Abril 2021].
- [44] D. Mastromatteo, «The Python pickle Module: How to Persist Objects in Python,» [En línea]. Available: <https://realpython.com/python-pickle-module/>. [Último acceso: Abril 2021].

- [45] A. Agrawal, «Understanding Python pickling and how to use it securely,» 18 Noviembre 2014. [En línea]. Available: <https://www.synopsys.com/blogs/software-security/python-pickling/>. [Último acceso: Abril 2021].
- [46] Wikipedia, «Unicode,» 6 Mayo 2020. [En línea]. Available: <https://es.wikipedia.org/wiki/Unicode>. [Último acceso: Abril 2021].
- [47] D. Carpentry, «Trabajando con Pandas DataFrames en Python,» [En línea]. Available: <https://datacarpentry.org/python-ecology-lesson-es/02-starting-with-data/>. [Último acceso: Mayo 2021].
- [48] D. t. Fish, «How to Drop Rows with NaN Values in Pandas DataFrame,» 4 Junio 2020. [En línea]. Available: <https://datatofish.com/dropna/>. [Último acceso: Mayo 2021].
- [49] E. D. Science, «How to Handle Imbalanced Classes in Machine Learning,» [En línea]. Available: <https://elitedatascience.com/imbalance-classes>. [Último acceso: Mayo 2021].
- [50] s.-l. developers, «sklearn.utils.resample,» [En línea]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html>. [Último acceso: Mayo 2021].
- [51] A. Bracco, «Normalización de Texto en Español de Argentina,» Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba, Cordoba, Argentina.
- [52] S. Prabhakaran, «Lemmatization Approaches with Examples in Python,» 2 Octubre 2018. [En línea]. Available: <https://www.machinelearningplus.com/lemmatization-examples-python/>. [Último acceso: Mayo 2021].
- [53] Wikipedia, «Stemming,» [En línea]. Available: <https://es.wikipedia.org/wiki/Stemming>. [Último acceso: Mayo 2021].
- [54] W. Cukierski, «The Enron Email Dataset,» 2016. [En línea]. Available: <https://www.kaggle.com/wcukierski/enron-email-dataset>. [Último acceso: Abril 2021].
- [55] namecheap, «What is an index page?,» 23 Enero 2018. [En línea]. Available: <https://www.namecheap.com/support/knowledgebase/article.aspx/183/27/what-is-an-index-page/>. [Último acceso: Mayo 2021].
- [56] M. contributors, «HTML basics,» 19 Febrero 2021. [En línea]. Available: [https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/HTML\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics). [Último acceso: Mayo 2021].
- [57] wikies, «nic.ly,» [En línea]. Available: <https://wikies.wiki/wiki/en/.ly>. [Último acceso: Mayo 2021].
- [58] V. Shah, «Hands-on implementation of TF-IDF from scratch in Python,» 29 Diciembre 2020. [En línea]. Available: <https://analyticsindiamag.com/hands-on-implementation-of-tf-idf-from-scratch-in-python/>. [Último acceso: Mayo 2021].
- [59] Geeks for Geeks, «Using CountVectorizer to Extracting Features from Text,» 17 Julio 2020. [En línea]. Available: <https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/>. [Último acceso: Mayo 2021].

- [60] C. Handbook, «How to Read Email From Gmail Using Python 3,» [En línea]. Available: <https://codehandbook.org/how-to-read-email-from-gmail-using-python/>. [Último acceso: Mayo 2021].
- [61] J. K. Korpela, «What is the RFC 822 format for the email addresses?,» 31 Agosto 2013. [En línea]. Available: <https://stackoverflow.com/questions/18551707/what-is-the-rfc-822-format-for-the-email-addresses>. [Último acceso: Mayo 2021].
- [62] J. Brownlee, «What is a Confusion Matrix in Machine Learning,» 18 Noviembre 2016. [En línea]. Available: <https://machinelearningmastery.com/confusion-matrix-machine-learning/>. [Último acceso: Junio 2021].
- [63] sentence, «Macro VS Micro VS Weighted VS Samples F1 Score,» Abril 2018. [En línea]. Available: <https://stackoverflow.com/questions/55740220/macro-vs-micro-vs-weighted-vs-samples-f1-score>. [Último acceso: Junio 2021].
- [64] N. Liberman, «Decision Trees and Random Forests,» 27 Enero 2017. [En línea]. Available: <https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991>. [Último acceso: Junio 2021].
- [65] Mark, «How to decide when to use Naive Bayes for classification,» Noviembre 2015. [En línea]. Available: <https://discuss.analyticsvidhya.com/t/how-to-decide-when-to-use-naive-bayes-for-classification/5720>. [Último acceso: Junio 2021].
- [66] M. A. Masten y A. Poirier, «Identification of Treatment Effects under Conditional Partial Independence,» Estados Unidos, 2017.
- [67] N. Unidas, «OBJETIVOS DE DESARROLLO SOSTENIBLE,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>. [Último acceso: Mayo 2021].
- [68] N. Unidas, «SALUD Y BIENESTAR,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/health/>. [Último acceso: Mayo 2021].
- [69] N. Unidas, «EDUCACIÓN DE CALIDAD,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/education/>. [Último acceso: Mayo 2021].
- [70] N. Unidas, «TRABAJO DECENTE Y CRECIMIENTO ECONÓMICO,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/economic-growth/>. [Último acceso: Mayo 2021].
- [71] N. Unidas, «CIUDADES Y COMUNIDADES SOSTENIBLES,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/cities/>. [Último acceso: Mayo 2021].
- [72] N. Unidas, «INDUSTRIA, INNOVACIÓN E INFRAESTURAS,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/infrastructure/>. [Último acceso: Mayo 2021].
- [73] N. Unidas, «REDUCCIÓN DE LAS DESIGUALDADES,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/inequality/>. [Último acceso: Mayo 2021].

- [74] N. Unidas, «PAZ, JUSTICIA E INSTITUCIONES SOLIDAS,» [En línea]. Available: <https://www.un.org/sustainabledevelopment/es/peace-justice/>. [Último acceso: Mayo 2021].
- [75] S. Learn, «scikit-learn, Machine Learning in Pyhton,» [En línea]. Available: <https://scikit-learn.org/stable/index.html>. [Último acceso: Abril 2021].

# Anexo I. Planificación

A continuación, se va a mostrar la lista de tareas que se han desarrollado y que más adelante, en el diagrama de Gantt, se mostrará en qué intervalo de fechas (aproximadas) se planificó su realización.

## Enumeración de tareas

- Comprensión del problema y estudio del estado del arte.
  - Determinación de requisitos.
  - Revisión de bibliografía
- Diseño e implementación de la solución.
  - Diseño de la arquitectura.
  - Implementar código con las herramientas seleccionadas
- Elaboración de un corpus: con un conjunto de textos de entrenamiento y un conjunto de textos de prueba.
- Pruebas y corrección de errores.
  - Revisión del código implementado
  - Pruebas unitarias.
  - Pruebas de integración.
  - Pruebas del sistema.
- Elaboración de documentación.
  - Informes y memoria.
  - Presentación.

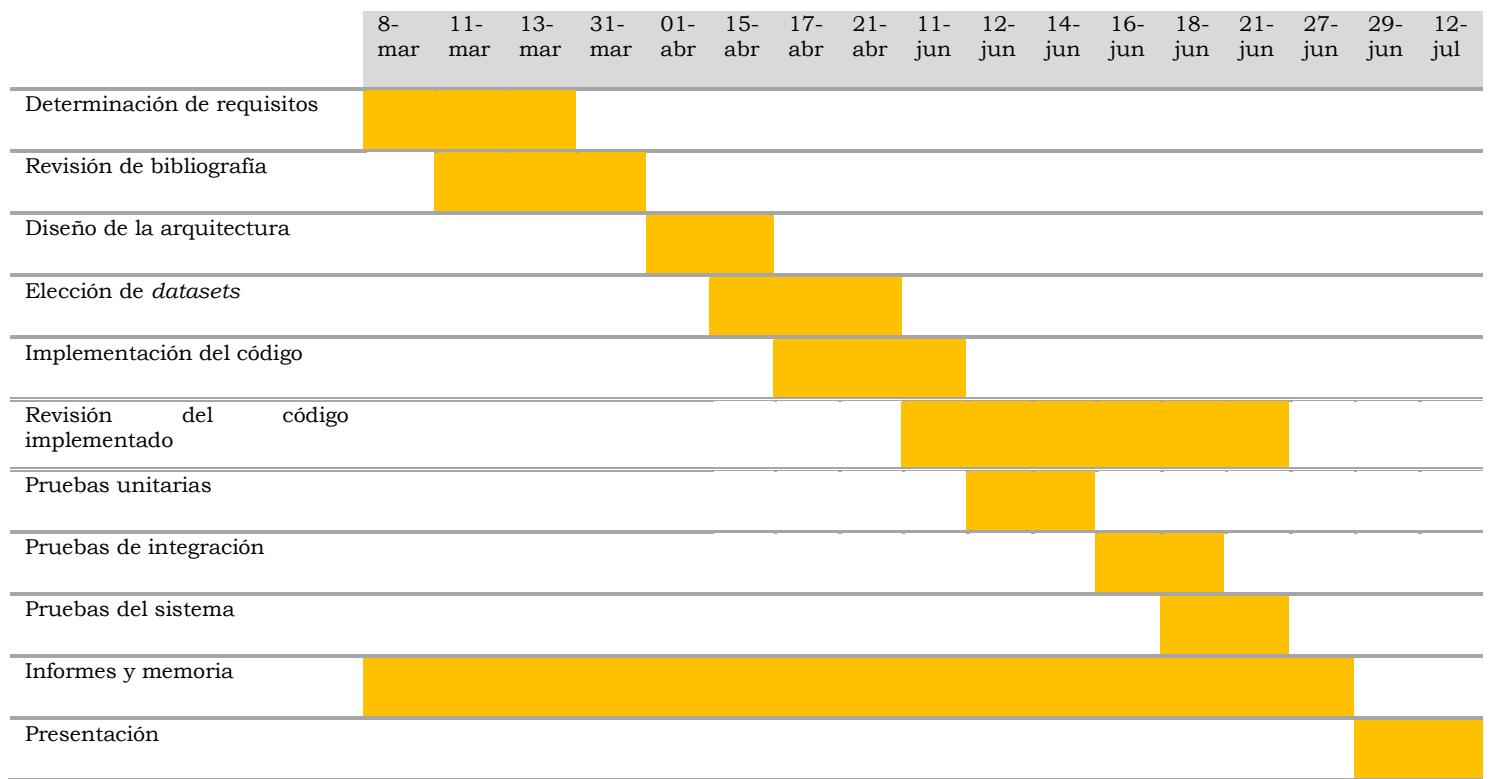
## Diagrama de Gantt

La *Tabla de anexo 1* recoge la lista de tareas junto con sus fechas de inicio y finalización.

Actividades	Inicio	Final	Horas	Tiempo en días
Determinación de requisitos	08-mar-21	13-mar-21	8	5
Revisión de bibliografía	11-mar-21	31-mar-21	90	20
Diseño de la arquitectura	01-abr-21	15-abr-21	20	14
Elección de <i>datasets</i>	15-abr-21	21-abr-21	5	6
Implementación del código	17-abr-21	11-jun-21	130	55
Revisión del código implementado	11-jun-21	21-jun-21	52	11
Pruebas unitarias	12-jun-21	14-jun-21	15	2
Pruebas de integración	16-jun-21	18-may-21	25	2
Pruebas del sistema	18-jun-21	21-jun-21	35	3
Informes y memoria	08-mar-21	27-jun-21	180	111
Presentación	29-jun-21	12-jul-21	10	14

*Tabla de anexo 1: Planificación de la lista de tareas*

La *Tabla de anexo 2* es el diagrama de Gantt correspondiente a las tareas de la *tabla 1*, tal como se han planificado finalmente.



*Tabla de anexo 2: Diagrama Gantt de la planificación de tareas*

## Anexo II. Descripción de la librería Scikit-learn

*Scikit-Learn* [75] es la librería que he utilizado para implementar mis algoritmos de *machine learning*. Hay que tener en cuenta que, para usarla, es necesario tener conocimientos como el ajuste de modelos, predicciones, o la validación cruzada. Es una librería que soporta tanto las técnicas para el aprendizaje supervisado como el no supervisado, y también proporciona herramientas varias como pueden ser el ajuste de modelos, el preprocesamiento de datos, o la selección de un modelo y su evaluación, entre muchos otros.

### Ajustando y prediciendo: bases del estimador

*Scikit-learn* proporciona decenas de algoritmo y modelos de *machine learning* construidos internamente, llamados estimadores. Cada estimador puede ser ajustado a cualquier conjunto de datos utilizando un método de ajuste.

En la *Tabla de anexo 3*, se muestra un ejemplo en el que ajustamos un clasificador de bosques aleatorios a unos datos básicos.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
X = [[ 1,  2,  3], # 2 muestras, 3 características
      [11, 12, 13]]
y = [0, 1] # clases de cada muestra
clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

*Tabla de anexo 3: Ejemplo de la clasificación de bosque aleatorio con datos básicos*

El método de ajuste acepta generalmente 2 inputs:

- La matriz de muestras (o matriz de diseño)  $X$ . El tamaño de  $X$  es típicamente  $(n_{muestras}, n_{características})$ , lo que significa que las muestras se representan en filas, y las características en columnas.
- El valor objetivo, que son números reales para las tareas de regresión, o enteros para las de clasificación (o cualquier otro conjunto de valores discretos). Para el aprendizaje no supervisado, y no tiene porque ser especificada.  $y$  es usualmente un array de dimensión 1 donde la  $i$ -enésima entrada corresponde al objetivo de la muestra  $i$ -enésima de la muestra (fila) de  $X$ .

Se espera que ambos  $X$  e  $y$  sean valores de tipo de arrays *numpy*, o cualquier otro valor similar de tipo de datos de array, aunque algunos estimadores funcionan con otros formatos como pueden ser matrices dispersas.

Una vez el estimador ha sido ajustado, puede ser usado para predecir valores objetivo de nuevos datos. No hace falta reentrenar al estimador, como se muestra en el código de la *Tabla de anexo 4*.

```
clf.predict(X) # predice clases de los datos de entrenamiento
array([0, 1])
clf.predict([[4, 5, 6], [14, 15, 16]]) # predice clases de los datos nuevos
array([0, 1])
```

*Tabla de anexo 4: Ejemplo de predicción de datos usando scikit-learn*

## Transformadores y preprocesadores

Los flujos de trabajo de *machine learning* normalmente se componen de varias partes. Una tubería típica tiene una etapa de preproceso que transforma o imputa los datos, y un predictor final que predice los valores objetivo.

En *scikit-learn*, los preprocesadores y transformadores siguen la misma API que los objetos estimadores (ambos heredan todo de la misma clase *BaseEstimator*). Los objetos transformadores no tienen un método de predicción, pero si un método de transformación de la que se obtiene una nueva matriz muestra  $X$  transformada, como se ve en la Tabla de anexo 5.

```
from sklearn.preprocessing import StandardScaler
X = [[0, 15],
      [1, -10]]
# escala los datos de acuerdo a los valores escalares computados
StandardScaler().fit(X).transform(X)
array([[-1.,  1.], [ 1., -1.]])
```

Tabla de anexo 5: Ejemplo de transformación de la matriz  $X$

A veces, es necesario aplicar diferentes transformaciones a diferentes características: el método *ColumnTransformer* está diseñado para estos casos.

## Tuberías: Encadenamiento de preprocesadores y estimadores

Los transformadores y estimadores (predictores) pueden ser combinados en un solo objeto unificado: un *Pipeline*. Este *pipeline* o tubería ofrece la misma API que un estimador normal: puede ser ajustado y usado para la predicción con las funciones *fit* y *predict*. Además, usando tuberías se puede prevenir la fuga de datos, como pueda ser el revelado de los datos de prueba en los datos de entrenamiento.

En el ejemplo de la *Tabla de anexo 6* se carga un data set de colores de iris de los ojos, dividido en datos de entrenamiento y de prueba, y se computa la marca de exactitud de una tubería en los datos de prueba.

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
...
# crea un objeto tubería
tuberia = make_pipeline(
    StandardScaler(),
    LogisticRegression()
)

# carga el dataset de iris y lo divide en datos de entrenamiento y prueba
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# ajusta toda la tubería de datos
tuberia.fit(X_train, y_train)
Pipeline(steps=[('standardscaler', StandardScaler()),
               ('logisticregression', LogisticRegression())])
```

```
# ahora se puede usar como cualquier otro estimador  
accuracy_score(tuberia.predict(X_test), y_test)
```

Tabla de anexo 6: Ejemplo del uso de tuberías con un data set del color de los ojos

## Evaluación de modelos

Ajustar un modelo a algunos datos no implica que vaya a predecir bien los datos que no se ven. Estos necesitan ser evaluados directamente. Se ha visto que el método auxiliar `train_test_split` proporciona muchas otras herramientas para la evaluación de modelos, en particular para la validación cruzada.

En el ejemplo de la *Tabla de anexo 7* se ve la implementación de un procedimiento de 5 pliegues de validación cruzada, usando la función auxiliar `cross_validate`. Nótese que es posible iterar manualmente a través de los distintos pliegues, usando distintas estrategias de separación de datos, y usar funciones de puntaje personalizadas.

```
from sklearn.datasets import make_regression  
from sklearn.linear_model import LinearRegression  
from sklearn.model_selection import cross_validate  
  
X, y = make_regression(n_samples=1000, random_state=0)  
lr = LinearRegression()  
  
result = cross_validate(lr, X, y) # CV predeterminado de 5 capas  
result['test_score'] # el rendimiento r-cuadrado es alto debido a que es un dataset sencillo  
array([1., 1., 1., 1., 1.])
```

Tabla de anexo 7: Ejemplo de validación cruzada para 5 pliegues

## Búsqueda de parámetros automática

Todos los estimadores tienen parámetros (normalmente llamados hiperparámetros) que se pueden rotar. El poder de generalización de un estimador casi siempre depende críticamente de unos pocos parámetros. Usualmente no se tienen claro los valores exactos de estos parámetros, ya que dependen de en los datos a mano.

*scikit-learn* proporciona herramientas que encuentran la mejor combinación de parámetros automáticamente (a través de la validación cruzada). En el ejemplo de la *Tabla de anexo 8* se busca aleatoriamente en el espacio de parámetros de un bosque aleatorio con un objeto `RandomizedSearchCV`. Cuando la búsqueda se ha terminado, el objeto `RandomizedSearchCV` se comporta como un regresor de bosques aleatorios que ha sido ajustado con el mejor conjunto de parámetros.

```
from sklearn.datasets import fetch_california_housing  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model_selection import RandomizedSearchCV  
from sklearn.model_selection import train_test_split  
from scipy.stats import randint  
  
X, y = fetch_california_housing(return_X_y=True)  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```

# definir el espacio de parametros sobre el que se buscará
param_distributions = {'n_estimators': randint(1, 5),
                      'max_depth': randint(5, 10)}

# se crea un objeto searchCV y se ajusta a los datos
search = RandomizedSearchCV(estimator=RandomForestRegressor(random_state=0),
                             n_iter=5,
                             param_distributions=param_distributions,
                             random_state=0)

search.fit(X_train, y_train)
RandomizedSearchCV(estimator=RandomForestRegressor(random_state=0), n_iter=5,
param_distributions={'max_depth': ...,
                     'n_estimators': ...},
                     random_state=0)

search.best_params_
{'max_depth': 9, 'n_estimators': 4}

# el objeto de búsqueda actúa ahora como un estimador de bosque aleatorio normal con una
# profundidad_máxima = 9 y n_estimadores = 4
search.score(X_test, y_test)

```

*Tabla de anexo 8: Ejemplo de búsqueda de parámetros automática con la regresión de bosques aleatorios*