UNIVERSIDAD
DE GRANADA

Facultad de Ciencias. Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicación

Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Técnicas de minería en bases de conocimiento

Presentado por:
Mónica Calzado Granados

Curso académico 2023-2024

Técnicas de minería en bases de conocimiento

Mónica Calzado Granados

Mónica Calzado Granados *Técnicas de minería en bases de conocimiento*.
Trabajo de fin de Grado. Curso académico 2023-2024.

Responsable de tutorización

Úrsula Torres Parejo
Departamento de Estadística e Investigación Operativa

Daniel Sánchez Fernández
Departamento de Ciencias de la Computación e Inteligencia Artificial

Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias.
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Universidad de Granada

Declaración de originalidad

D./Dña. Mónica Calzado Granados

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2023-2024, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 12 de julio de 2024

Fdo: Mónica Calzado Granados

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Agradecimientos (opcional, ver archivo `preliminares/agradecimiento.tex`).

Resumen

En este Trabajo de Fin de Grado se destaca la creciente importancia de la Minería de Texto y el papel crítico que juegan las bases de conocimiento en el análisis de datos textuales. Un texto puede considerarse como una base de conocimiento articulada en lenguaje natural, lo que permite la aplicación de métodos analíticos avanzados para extraer información valiosa que podemos convertir en conocimiento. A pesar de la disponibilidad de numerosos enfoques innovadores en el ámbito de la Minería de Texto y la Ingeniería de Conocimiento, en este TFG se explotan técnicas más convencionales. Este trabajo busca explorar y aplicar métodos existentes para mejorar la comprensión y el procesamiento del texto.

El problema específico que abordamos es la detección de *spam*, dada su relevancia en el contexto actual, donde la comunicación digital juega un papel esencial en nuestra sociedad. La detección eficaz de *spam* no solo puede mejorar la experiencia del usuario, sino que también protege contra posibles amenazas y fraudes en línea. El estudio de corpus textuales, especialmente en aplicaciones como el filtrado de spam, representa un desafío en sí mismo, debido, en primer lugar, a la necesidad de procesar y analizar grandes volúmenes de datos no estructurados de manera eficiente; y en segundo lugar, al continuo desarrollo y mejora de los métodos utilizados por los *spammers* para conseguir sus objetivos.

En este contexto, se exploran las bases matemáticas de varios algoritmos de aprendizaje supervisado, con especial atención en los clasificadores Naive Bayes y las Máquinas de Soporte Vectorial (SVM). Se ha elegido esta combinación de enfoques debido a la eficacia demostrada de ambos en tareas de clasificación de texto, y a que abordan el problema de clasificación desde dos perspectivas complementarias: una probabilística y otra geométrica. Esto permite un enfoque comparativo para identificar la técnica más adecuada bajo diversas condiciones experimentales.

La metodología seguida en la fase de experimentación del trabajo sigue las distintas fases extraídas del proceso conocido como *Knowledge Discovery in Text* (KDT). La finalidad de seguir este proceso es la de conseguir el máximo desempeño posible de los modelos entrenados.

Inicialmente, se realiza una limpieza exhaustiva de los datos para eliminar inconsistencias y preparar el texto para su análisis. Al trabajar con datos textuales, estos se procesan de forma diferente a datos estructurados; la preparación de estos datos incluye técnicas de tokenización, eliminación de *stop words* y términos numéricos, lematización, etc.

A continuación, se lleva a cabo un análisis exploratorio de los datos con el fin de encontrar patrones y tendencias que puedan aportarnos una comprensión más profunda sobre este. Este análisis incluye, por un lado, la visualización de características más superficiales mediante, por ejemplo, el uso de histogramas de la longitud de los textos o el conteo de palabras. Por otro lado, estudiamos con más detalle el contenido de los textos a través de *word clouds* y n-gramas.

Luego, se realiza una transformación de los datos para facilitar su tratamiento. Seguidamente, se adopta una estrategia adecuada para la representación de los datos, utilizando

técnicas de vectorización como TF-IDF y conteo de frecuencias, que transforman el texto en un formato adecuado para el procesamiento de los algoritmos. Los algoritmos de aprendizaje automático se aplican posteriormente para evaluar y comparar el rendimiento entre ellos. Además, se lleva a cabo una exhaustiva búsqueda de hiperparámetros para optimizar cada modelo.

Adicionalmente, para completar el proyecto, se intenta trabajar con nuevos enfoques de Procesamiento del Lenguaje Natural, como es estudiar la relación semántica entre los términos que componen los textos. Mediante *synsets* o conjuntos de sinónimos de un término, podemos establecer una relación con otros términos que en un principio un modelo convencional no podría detectar. A través de librerías como `NLTK` se transforma nuestro corpus textual en uno con términos más generales conocidos como hiperónimos. De nuevo se estudia el desempeño de los algoritmos seleccionados y se compara con el de los modelos anteriores.

Por último, se exponen los resultados y las conclusiones extraídas de esta experimentación. Veremos, por ejemplo, que hay términos concretos en los correos que pueden ser más determinantes que otros a la hora de decidir si se trata de *spam* o no. También se habla de otras técnicas e ideas nuevas que podríamos aplicar al problema en trabajos futuros, extraídas del presente estudio.

Palabras clave: minería de texto, aprendizaje automático, bases de conocimiento, *spam*, clasificación, red bayesiana, máquinas de soporte vectorial, análisis exploratorio.

Summary

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.tex`

Introducción

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.

Ver archivo `preliminares/introduccion.tex`

1. Minería de Textos

La Minería de Textos es una rama de estudio crucial en la Ciencia de Datos. Se ha convertido en una herramienta invalorable para extraer conocimiento a partir de grandes cantidades de texto no estructurado.

En los últimos años, la Minería de Textos ha ganado una gran importancia debido al explosivo crecimiento de datos textuales que se pueden encontrar en diversas fuentes: documentos, correos electrónicos, redes sociales, páginas web y otros tipos de contenido textual.

Esta disciplina se ha vuelto esencial en la medida en que se encarga de analizar y descubrir nuevos patrones, tendencias y relaciones significativas dentro de conjuntos de información no estructurada. Mediante el uso de técnicas avanzadas de Procesamiento del Lenguaje Natural (PLN) y Aprendizaje Automático (en inglés *Machine Learning*), la Minería de Textos permite a las empresas, científicos y organizaciones obtener información valiosa que les ayuda a impulsar una toma de decisiones estratégica.

1.1. *Knowledge Discovery from Databases* (KDD)

La información contenida en las bases de datos puede ser analizada de forma manual por expertos para obtener conclusiones. Sin embargo, en estos tiempos de avance tecnológico, debido al gran volumen de datos, necesitamos encontrar una forma de automatizar el proceso.

Surge así el Descubrimiento de Conocimiento en Bases de Datos (en inglés KDD, *Knowledge Discovery from Databases*), que tiene sus bases en disciplinas como la Estadística, Sistemas de Información y Aprendizaje Automático. El término KDD se puede definir como:

> *El proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y, en última instancia, comprensibles a partir de los datos* [1].

1.1.1. Fases del KDD

El proceso de KDD resulta bastante interactivo, en el sentido de que intervienen muchas decisiones tomadas por el usuario según este considere oportuno. Estos pasos se pueden resumir de la siguiente manera [1]:

1. **Aprendizaje sobre el dominio de la aplicación:** es necesario determinar los objetivos y reunir el conocimiento previo que tenemos sobre el problema.

2. **Creación de un corpus de datos:** seleccionamos un conjunto de datos o *dataset* del cual queremos extraer conocimiento.

3. **Limpieza de datos y preprocesamiento:** partiendo de los datos recopilados, esta es la fase en la que se eliminan los datos ruidosos y anómalos. También se llevan a cabo estrategias para manejar valores perdidos.

4. **Reducción de datos:** consiste en seleccionar las características más útiles para representar los datos, y reducir la dimensionalidad del problema descartando atributos poco relevantes.

5. **Elegir una técnica de minería de datos:** se trata de decidir el propósito del modelo (qué tipo de información se quiere descubrir) y qué técnica de minería de datos podemos aplicar para ello (sumarización, clasificación, regresión, clustering, etc).

6. **Elegir un algoritmo de minería de datos:** incluye decidir qué modelos y parámetros pueden ser apropiados (por ejemplo, existen modelos para datos categóricos que son distintos de los modelos para datos numéricos).

7. **Minería de datos:** consiste en el descubrimiento de patrones, relaciones y conocimiento de interés, mediante el uso de algoritmos de clustering, clasificación, regresión, asociación, y otras técnicas de Aprendizaje Automático.

8. **Interpretación:** evaluar los patrones descubiertos y regresar a cualquiera de los pasos anteriores si es necesario. También trata la representación y visualización de los patrones extraídos, para facilitar la comprensión por los usuarios.

9. **Utilización del conocimiento descubierto:** consiste en acciones desde incorporar el nuevo conocimiento en un sistema inteligente, llevar a cabo acciones respaldadas en dicho conocimiento, o simplemente documentarlo y contrastarlo con conocimiento anteriormente extraído.

La Figura 1.1 muestra un resumen de las fases del proceso KDD, donde podemos apreciar que aunque el proceso es lineal, se puede regresar a cualquiera de las fases anteriores si se considera necesario.

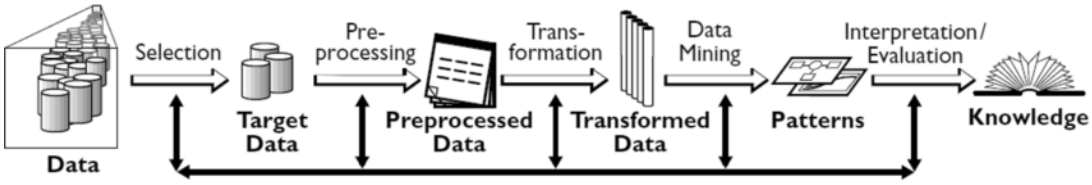


Figura 1.1.: Fases del proceso KDD [1]

La mayoría de los estudios sobre KDD se enfocan principalmente en la fase de Minería de Datos. Sin embargo, los otros pasos son igualmente importantes para la aplicación del proceso de KDD.

1.2. Knowledge Discovery from Text (KDT)

La mayoría de información generada por las organizaciones se encuentra en forma de texto. Procesar dicha información puede resultar difícil debido al gran volumen de documentos y la falta de estructura de estos con respecto a una base de datos convencional. Es por ello que necesitaremos tratar algunas de las fases del KDD de forma distinta al tratar con textos.

1.2.1. KDD vs KDT

En general, la diferencia principal entre KDD y KDT reside en que el texto carece de una estructura procesable de forma automática. En tal caso, necesitamos de un proceso previo de transformación del texto en datos, utilizando para ello diversas estructuras de datos, a las que llamaremos formas intermedias. Por tanto, en el KDT las etapas de selección, preprocesamiento y transformación de los datos textuales se convierten en fundamentales e imprescindibles [9]. A partir de ahora nos referiremos a esta etapa como Preprocesamiento. En la tabla 1.1, inspirada en [9], se muestra una comparación entre las fases de KDD y su equivalencia en el proceso KDT. Podemos observar que algunas fases se mantienen igual en ambos casos y otras difieren en el enfoque y las herramientas utilizadas:

Fase	KDD (Knowledge Discovery in Databases)	KDT (Knowledge Discovery in Texts)
1	Comprender el dominio de la aplicación.	Definición de los conceptos de interés y determinar un objetivo.
2	Seleccionar un conjunto de datos objetivo. Generar una base de datos o tomar una.	Los textos se obtienen con herramientas de Recuperación de Información o de forma manual.
3	Limpieza (eliminar ruido) y manejar valores perdidos.	Preparar texto a un formato aceptado por las técnicas de minería (formas intermedias)
4	Transformación de datos (reducción de dimensionalidad, tareas de balanceo de clases...)	Transformación de datos: se eliminan stop words y palabras frecuentes poco relevantes para reducir la dimensionalidad del problema, tokenización, etc. Utilización de redes semánticas y otras herramientas para agrupar conceptos.
5	Elegir una técnica adecuada de minería de datos.	Elegir una técnica adecuada de minería de textos.
6	Elección de algoritmos de minería de datos. Elegir parámetros apropiados.	Elección de algoritmos de minería de datos que funcionen bien con texto. Elegir parámetros apropiados.
7	Aplicación del algoritmo	Aplicación del algoritmo
8	Interpretar patrones descubiertos. Cálculo de medidas de validación. Visualización.	Visualización (nubes de palabras, ocurrencia de términos...) para interpretar los resultados y encontrar nuevos patrones en el texto.
9	Utilización del nuevo conocimiento.	Utilización del nuevo conocimiento.

Tabla 1.1.: Comparativa entre los procesos KDD y KDT

1.2.2. Fases del KDT

Teniendo en cuenta las comparaciones de la tabla 1.1, observamos que el proceso de KDT utiliza sus propias tareas de descubrimiento de conocimiento, que a veces coinciden con las del KDD y otras no. De forma más general, podemos dividir las fases del KDT en tres [9]:

- **Preprocesamiento:** antes de realizar la minería, necesitamos preparar el texto a un formato que sea aceptado por las técnicas que queremos aplicar. Esto abarca desde la homogeneización de distintos documentos, hasta la representación en Formas Intermedias. Entre las distintas técnicas de transformación de texto encontramos las siguientes:
  1. Normalización de texto: convertir todo el texto a minúsculas o mayúsculas para que las palabras sean tratadas de manera uniforme.
  2. Tokenización: dividir el texto en unidades más pequeñas, como palabras o frases, llamadas tokens.
  3. Etiquetado: asignar a cada token una etiqueta que indica su función gramatical: sustantivo, verbo, adjetivo, etc.
  4. Eliminación de *stop words*: eliminar palabras comunes pero no informativas que no aportan significado al texto, como determinantes, preposiciones, etc.
  5. *Stemming*: Reducir las palabras a su raíz, eliminando sufijos y prefijos, para reducir las palabras a su forma base.
  6. Corrección ortográfica: Corregir errores ortográficos comunes para mejorar la precisión del análisis.
  7. Eliminación de palabras raras o infrecuentes: Eliminar palabras que aparecen con poca frecuencia en el corpus, ya que pueden no ser útiles para el análisis.
  8. Eliminación de URLs, referencias y etiquetas HTML, que no aportan información semántica.
  9. Uso de *synsets* [10] que identifican conceptos a partir de palabras sinónimas, haciendo uso de diccionarios ontológicos.

- **Minería de textos**: En la literatura, algunos autores identifican el término *Minería de Textos* con todo el proceso de Descubrimiento de Conocimiento en Texto (KDT) [9]; sin embargo, en este contexto, nos referiremos específicamente a la fase encargada de localizar patrones, relaciones y estructuras interesantes, útiles y desconocidos en el texto.

- **Visualización**: una vez obtenidos los resultados de la fase anterior, es conveniente una fase de visualización para representar de manera efectiva los patrones, términos, tendencias y los resultados descubiertos. La visualización juega un papel crucial en la comprensión de la información extraída, permitiendo a los usuarios explorar y analizar los datos de manera intuitiva y amigable.
  Se pueden emplear numerosas herramientas dependiendo de la técnica de minería empleada, como nubes de palabras, diagramas de ocurrencia de términos, gráficos de dispersión de términos, clusterización de documentos, etc. Estas herramientas no sólo facilitan la interpretación de los resultados, sino que también pueden ayudar a descubrir más patrones, relaciones semánticas entre términos, etc.

1.2.3. Aplicaciones del KDT

Debido a la dificultad inherente de estructurar y procesar el texto de manera adecuada, las aplicaciones de la Minería de Texto enfrentan desafíos bastante complejos. Al igual que en la Minería de Datos, los expertos humanos tienen un papel crucial a la hora de interpretar

los resultados de manera efectiva. Entre las distintas aplicaciones del KDT destacamos las siguientes [9]:

1. **Segmentación de clientes:** se trata de clasificar a los clientes en grupos según patrones de compra, preferencias o características demográficas. Esta segmentación permite a las empresas satisfacer mejor las necesidades del cliente y realizar estrategias de Marketing más eficaces.

2. **Detección de fraude financiero:** este es un desafío constante en el sector financiero debido a la evolución de las técnicas empleadas por los delincuentes. Existen algoritmos de detección de anomalías, que se utilizan para detectar transacciones y patrones de compras inusuales para un cliente específico.

3. **Filtrado de correos electrónicos:** este problema surge debido al volumen considerable de mensajes que un usuario puede recibir diariamente, lo que puede resultar abrumador sin herramientas para gestionarlos. La implementación de filtros de correo electrónico puede mejorar significativamente la productividad en el lugar de trabajo, al ayudar a procesar los correos de forma más eficiente.

4. **Análisis de sentimientos:** se trata de una técnica fundamental para comprender la opinión y el comportamiento de los usuarios en plataformas como Twitter, Facebook o Instagram. Esto implica la extracción de tendencias y patrones a partir de las publicaciones y comentarios hechos por los usuarios. En el ámbito político, el análisis de sentimientos en las redes sociales ha demostrado ser útil para evaluar la popularidad de los candidatos y comprender las preocupaciones y necesidades de los votantes. Esta información puede resultar crucial para diseñar con éxito campañas políticas.

5. **Medicina personalizada:** es un área revolucionaria en la atención médica cuyo objetivo es adaptar los tratamientos y terapias a las características individuales de cada paciente. Para ello se nutre de datos genéticos y clínicos para identificar patrones que ayuden a predecir la reacción del paciente ante un tratamiento específico.

1.3. Procesamiento del lenguaje natural

El Procesamiento del Lenguaje Natural (PLN) es un campo de estudio centrado en el desarrollo de sistemas que puedan analizar y comprender el lenguaje humano a partir de un texto libre.

El objetivo del PLN es extraer el significado y el contexto de los textos. Usando PLN podemos hacer tareas como resumen automático de textos, generación de textos, traducción de idiomas, análisis de sentimientos, reconocimiento del habla y clasificación de artículos por temáticas. Para ello, utiliza conceptos lingüísticos como nombres, verbos, adjetivos, etc. El PLN se enfrenta a retos como las numerosas ambigüedades que contiene el lenguaje natural, tanto de palabras con doble significado como de estructuras gramaticales o frases hechas.

Para llevar a cabo estas tareas, el PLN hace uso de *representaciones de conocimiento*, como diccionarios de palabras con su significado, conjuntos de reglas gramaticales, etc. También se suele hacer uso de ontologías, diccionarios de sinónimos, abreviaciones, etc.

El Procesamiento del Lenguaje Natural es un paso clave en el proceso de extracción de conocimiento a partir de texto en el KDT. En este contexto, el PLN se utiliza para transformar

el texto en una representación estructurada y procesable, permitiendo así aplicar técnicas de minería de texto. Algunas tareas típicas del PLN en el KDT incluyen la tokenización (división del texto en unidades más pequeñas como palabras o frases), el etiquetado de partes del discurso, la eliminación de stopwords, la detección de entidades nombradas, el análisis de sentimientos y la extracción de información relevante [11].

1.4. Formas intermedias

Debido a que el texto contiene información no estructurada, dentro de la etapa de Prepocesamiento del proceso KDT, es crucial convertir el texto a una representación o forma intermedia que se pueda manejar de forma computacional. Las posibilidades incluyen desde dividir el texto en unidades mínimas hasta identificar relaciones semánticas entre palabras. Al mismo tiempo se debe procurar no perder la integridad del texto inicial, es decir, no perder información sobre las relaciones entre términos. A continuación veremos algunas de las formas intermedias comúnmente empleadas [9].

1.4.1. Palabra

La palabra es el elemento mínimo de estudio sobre un texto [9]. Por sí misma, una palabra no aporta suficiente información, pero algunas consideraciones como el conteo de palabras y la riqueza léxica (proporción de palabras diferentes con respecto al total) pueden proporcionar información sobre el texto.

1.4.2. Bolsa de palabras

Consiste en recopilar todas las palabras que aparecen en un documento, ignorando el orden, la estructura gramatical y las relaciones semánticas. En esta representación, cada palabra se considera un *token* y se crea un vector que indica cuántas veces aparece cada palabra en el documento. Dentro de una bolsa podemos encontrar distintos tipos de tokens [9]:

- **Palabra vacía** *(stop word)*: preposiciones, artículos y conjunciones, en definitiva son palabras que sirven para relacionar términos pero por sí mismas no aportan valor.
- **Palabra clave** *(keyword)*: se trata de palabras que identifican la temática del texto. Se les suele asociar un peso numérico para destacar su importancia.

A modo de ejemplo, podemos considerar el siguiente texto:

El perro persigue al gato. El gato huye del perro. (1.4.2)

La representación del texto mediante una bolsa de palabras podría verse como en la tabla 1.2. La misma bolsa de palabras pero eliminado las *stop words*, se vería como en la tabla 1.3.

Observamos que al representar el texto como una bolsa de palabras, se pierde completamente el orden. La bolsa de palabras nos dice cuántas veces aparece cada palabra, pero ya no sabemos nada sobre la relación entre el perro y el gato, y entre quién persigue y quién huye. Esta pérdida de contexto puede ser crítica en algunas tareas, ya que al perder la relación entre palabras, podemos perder información crucial.

Palabra	Frecuencia
el	2
perro	2
persigue	1
al	1
gato	2
huye	1
del	1

Tabla 1.2.: Bolsa de palabras del ejemplo 1.4.2

Palabra	Frecuencia
perro	2
persigue	1
gato	2
huye	1

Tabla 1.3.: Bolsa de palabras sin *stop words* del ejemplo 1.4.2

1.4.3. Concepto

Un concepto es la representación mental de un objeto o una idea, expresada a través de un término [12]. Los conceptos organizan palabras relacionadas bajo una idea común. Por ejemplo, el concepto de *animales domésticos* podría incluir palabras como perro, gato, pájaro, etc. Uno de los inconvenientes al tratar con conceptos es que a diferencia de las palabras, estos suelen depender del contexto.

1.4.4. Taxonomía

Una taxonomía o jerarquía de conceptos es una clasificación que organiza de forma ordenada conceptos o términos que se encuentran en niveles jerárquicos distintos, basándose en las relaciones entre ellos [13]. Las taxonomías pueden representarse mediante jerarquías de términos, donde los conceptos más generales aparecen en los niveles superiores, y los conceptos más particulares aparecen en niveles inferiores. Un ejemplo de taxonomía del reino animal es el siguiente:

1. *Animalia*
    a) *Mammalia*
        I. *Felidae (Gatos)*
        II. *Canidae (Perros)*
    b) *Aves*
        I. *Accipitridae (Águilas)*
        II. *Anatidae (Patos)*
    c) *Reptilia*
        I. *Pythonidae (Pitones)*

II. *Crocodylidae (Cocodrilos)*

1.4.5. Grafos

Un grafo o red semántica es una representación visual de conocimiento que emplea conceptos y las relaciones semánticas entre ellos [9]. Estos conceptos son representados mediante nodos, y las relaciones mediante aristas o enlaces.

Una característica distintiva de las redes semánticas es el uso de la herencia, donde los nodos hijos heredan las características de los nodos padres. Estas redes son útiles para representar y comprender la semántica y las relaciones entre conceptos en un dominio específico. Además, destacan por ser capaces de preservar gran carga semántica, mientras que otras formas intermedias la pierden. Sin embargo, en ocasiones esto se traduce en un aumento excesivo de la complejidad del tratamiento de datos.

Entre los distintos tipos de red semántica destacan las redes IS-A y los grafos conceptuales [14].

1.4.6. Ontología

La literatura científica ofrece una gran variedad de definiciones para las ontologías. En términos generales, se trata de un mapa conceptual detallado y estructurado del conocimiento sobre un dominio específico. Una ontología está formada por los siguientes elementos básicos [9]:

1. *Conceptos*: representan las entidades o clases dentro del dominio de interés. Por ejemplo, en una ontología sobre automóviles, podría haber conceptos como *Automóvil*, *Motor*, *Rueda*, etc.
2. *Roles*: describen relaciones binarias entre conceptos y propiedades. Por ejemplo, las propiedades *color*, *velocidad máxima*, *número de puertas* y las relaciones *es un tipo de*, *tiene*, etc.
3. *Individuos*: instancias o ejemplos.
4. *Axiomas*: especifican las reglas o restricciones que deben cumplirse en el dominio. Por ejemplo, *un automóvil tiene cuatro ruedas*.

Las ontologías proporcionan un marco formal para representar el conocimiento, lo que facilita la interoperabilidad entre distintos sistemas y permite a las máquinas realizar inferencias y razonamientos sobre el dominio de conocimiento especificado.

1.5. Minería de bases de conocimiento

En la minería de datos, el conocimiento es obtenido mediante razonamiento de tipo *inductivo*. Consiste en encontrar patrones o modelos generales que expliquen los datos. Se trata de un proceso que va de lo particular a lo general, es decir, se parte de los datos para llegar a conclusiones sobre estos. Este es el tipo de razonamiento que se aplica en las técnicas de clustering, reglas de asociación, árboles de decisión, etc. [9]

La minería de textos puede verse como un caso particular de la minería de datos, donde es necesario un proceso previo de transformación del texto en datos (formas intermedias), dado que el lenguaje natural de un texto es incomprensible e improcesable por una máquina.

Este enfoque convencional ha demostrado ser muy útil en una gran cantidad de aplicaciones; sin embargo, no podemos olvidar que el texto contiene una capacidad expresiva mucho mayor que las estructuras de datos. Por tanto, al reducir el texto a forma intermedia, se pierde una gran cantidad de información valiosa. De aquí surge la necesidad de definir los términos *conocimiento* y *base de conocimiento* para hacer referencia a una representación del texto más rica que los datos y las bases de datos.

1.5.1. Bases de conocimiento

En el ámbito de Ingeniería del Conocimiento, se distinguen tres niveles de contenido: *datos*, *información* y *conocimiento* [15]:

- Los **datos** son la mínima unidad semántica, son hechos o elementos discretos que pueden ser registrados o almacenados. Por sí solos son irrelevantes y no aportan información.
- La **información** es el resultado del procesamiento de esos datos, lo que les otorga significado y contexto.
- El **conocimiento** va un paso más allá al incorporar la comprensión y la capacidad de aplicar la información en situaciones reales, por ejemplo para la toma de decisiones.

Las *bases de conocimiento* son estructuras que se utilizan para la representación del conocimiento de un dominio específico. En el contexto de la inteligencia artificial, una base de conocimiento se utiliza para capturar información relevante, reglas, hechos, conceptos y relaciones dentro de un área de interés. Una base de conocimiento puede estar formada por distintos tipos de datos estructurados, como ontologías, reglas de producción, redes semánticas, grafos, entre otros.

Teniendo esto en cuenta, la Ingeniería de conocimiento se enfoca en convertir datos en conocimiento accionable, mediante un proceso que usa métodos y técnicas para estructurar y procesar datos de manera que puedan ser analizados para tomar decisiones.

La Ingeniería de conocimiento y la Ciencia de datos tienen una relación estrecha en el ámbito de la inteligencia artificial. Mientras que la Ciencia de datos se centra en el estudio y análisis de grandes volúmenes de datos con el fin de obtener patrones y relaciones, la Ingeniería de conocimiento se enfoca en representar y utilizar ese conocimiento de manera efectiva.

1.5.2. Sistemas de representación de conocimiento

Hemos visto que una base de conocimiento es un sistema que almacena y organiza el conocimiento de un dominio específico para que pueda ser accesible por otros sistemas de software. Dentro de una base de conocimiento podemos encontrar información, hechos, reglas, y más elementos que ayudan a describir y comprender el dominio estudiado. En este contexto, una base de conocimiento puede estar representada mediante estructuras de datos tales como redes semánticas, marcos, reglas de producción, ontologías, entre otros. Es a estos elementos a lo que denominamos sistemas de representación de conocimiento.

En breves palabras, una estructura de representación de conocimiento es la estructura interna que define de qué forma está modelado y organizado el conocimiento dentro de una base de conocimiento. Como es de esperar, esta estructura debe ser fácil de manejar computacionalmente y útil de comprender tanto por humanos como por máquinas.

Existen varios tipos de sistemas de representación de conocimiento, algunos de los más comunes son [15]:

- **Redes semánticas**: representan el conocimiento de forma simple y visual, mediante un grafo dirigido etiquetado. Está formada por nodos (conceptos) y arcos (relaciones binarias entre los conceptos). Entre las relaciones que se suelen usar entre las redes semánticas, encontramos *Subclase-de* e *Instancia-de*, que sirven para representar la herencia entre conceptos. La herencia se corresponde con el razonamiento de que *las subclases y las instancias heredan las propiedades de las clases más generales*. Esta estructura se suele utilizar por ejemplo para representar relaciones de hiponimia e hiperonimia entre conceptos.

- **Marcos o *Frames***: Cada frame representa un objeto o un concepto relevante, e incluye las propiedades del mismo. Un conjunto de frames trata de representar el conocimiento de un dominio de interés, y está organizado de forma jerárquica en una taxonomía. Estos frames se dividen en frames *clase* (frames genéricas, representan conocimiento de clases de objetos) y frames *instancia* (representan conocimiento de objetos individuales). Esta estructura también sirve para representar herencia entre conceptos. Cada frame hereda las propiedades del frame padre. Por tanto los frames están relacionados con las redes semánticas en tanto que podemos representar una red semántica mediante un conjunto de frames.

- **Reglas de producción**: Se utilizan en sistemas expertos para representar el conocimiento de forma condicional. Consisten en conjuntos de reglas del tipo *Si-Entonces*, donde se especifica qué hacer si se cumple cierta condición. La estructura general de una regla de producción es la de *Antecedente* $\implies$ *Consecuente*, donde el antecedente contiene las cláusulas que deben cumplirse para aplicar la regla y el consecuente indica las conclusiones o acciones a realizar por el sistema.
  Como ejemplo de aplicación de las reglas de producción, podemos considerar un sistema para el diagnóstico médico, donde se diagnostique una patología en base a los síntomas del paciente. Una regla de este sistema podría ser *Si el paciente tiene fiebre y dolor de garganta, entonces diagnosticar una infección de garganta*.

- **Lógica de Predicados**: Es un formalismo matemático para representar el conocimiento mediante predicados y cuantificadores. Permite expresar proposiciones y relaciones de manera precisa, lo cual facilita el razonamiento lógico y la inferencia en bases de conocimiento.

- **Ontologías**: Como ya vimos en 1.4.6, son modelos conceptuales que representan un conjunto de conceptos y las relaciones entre ellos en un dominio específico. Gracias a las ontologías podemos definir términos y sus relaciones de manera formal, para facilitar el intercambio de datos entre varios sistemas informáticos.
  Las ontologías usan para la representación de conceptos e instancias y para el razonamiento lo que se conoce como *lógicas descriptivas*, que son distintos subconjuntos de la lógica de predicados que tienen características computacionales razonables, como la decidibilidad, entre otras.

1.5.3. Minería de textos y Minería de bases de conocimiento

La minería de bases de conocimiento se podría situar en un marco más general que el de la minería de textos, pues un texto puede verse como una representación de conocimiento con un formato de difícil procesamiento para razonar, pero que puede usarse como punto de partida para extraer conocimiento en forma de cualquiera de las representaciones intermedias que hemos nombrado en la sección 1.5.2.

Mientras que la minería de textos está principalmente enfocada en el análisis de grandes cantidades de textos sin formato, la minería de bases de conocimiento incluye cualquier sistema informático que contenga información valiosa. Esto puede incluir desde repositorios de información hasta sistemas basados en reglas y sistemas de gestión de documentos, entre otros.

1.5.4. Relación de las bases de conocimiento con el proyecto

En este proyecto, se llevará a cabo la implementación de los pasos previamente estudiados del proceso de KDT, con el objetivo de transformar una base de datos textual inicial en un formato que sea procesable por una máquina. Esto se hará en primer lugar mediante técnicas de preprocesamiento típicas del KDT. Además de esto, se podrá hacer uso de estructuras de conocimiento tales como ontologías y redes semánticas, que ayuden a estructurar el texto de forma más efectiva, teniendo ahora en cuenta los conceptos y las relaciones semánticas entre ellos.

Por ejemplo, mediante la detección de sinónimos dentro de un corpus, podríamos agrupar palabras que en un principio no tendrían relación entre sí si sólo aplicáramos las técnicas estándar de preprocesamiento. Además, mediante redes semánticas podemos agrupar distintas palabras si están vinculadas jerárquicamente mediante relaciones de hiponimia e hiperonimia.

Es por esto que hemos elegido el título *Minería de bases de conocimiento* en lugar de *Minería de bases de datos textuales*. Hemos considerado que el proceso de KDT abarca un marco más extenso que la simple aplicación de algoritmos de detección de patrones, y que con el respaldo de herramientas de Ingeniería de Conocimiento, podemos enriquecer el proceso de Descubrimiento de Conocimiento para lograr resultados más reveladores.

2. *Spam*

En este capítulo vamos a abarcar el concepto de *spam*, los tipos de *spam*, así como las distintas técnicas existentes en la literatura para detectarlo.

2.1. Definición e historia

El *spam* es el envío masivo y no solicitado de mensajes electrónicos, ya sea por correo electrónico, mensajes de texto, redes sociales y otros medios digitales. Generalmente los mensajes *spam* tienen como objetivo promocionar productos o servicios. A veces, estos mensajes también son usados para engañar a los usuarios para que revelen información personal, como contraseñas o información financiera.

En contraste con el spam, el término *ham* se refiere a los mensajes de correo electrónico legítimos y deseados, que son enviados por remitentes conocidos o esperados. Estos mensajes suelen ser correos electrónicos personales, comerciales o informativos que se envían a destinatarios que han optado por recibirlos, por lo que no presentan ningún tipo de amenaza o molestia.

El término *spam* en el contexto de los mensajes de correo electrónico, se popularizó en la década de los años 90. El primer mensaje de *spam* conocido tiene su origen 1994, en un post de Usenet, donde una firma de abogados publicó un mensaje de anuncio de su firma legal. Desde entonces, la publicidad y el marketing mediante correo electrónico ha crecido a niveles impensables [16].

Según cifras del informe Kaspersky [17], se estima que en 2022 el 48.63 % de correos electrónicos en todo el mundo fueron *spam*.

2.2. Tipos de *spam*

El *spam* no es solo molesto para los usuarios, sino que también puede presentar una amenaza y un riesgo para preservar la seguridad y la privacidad online. Los *spammers*, quienes envían estos mensajes no deseados, utilizan diversas técnicas para difundir dicho contenido no solicitado. A continuación vamos a exponer los distintos tipos de *spam* conocidos, según su naturaleza y el riesgo que representan [18]:

- **Correo electrónico no deseado:** es el tipo de *spam* más común y todos estamos habituados a encontrarlo. Con frecuencia inundan nuestra bandeja de entrada, resultando muy molesto al usuario. Tiene fines publicitarios o promocionales, por lo que no suelen representar un riesgo directo para la seguridad del usuario, aunque consumen recursos de almacenamiento.

- ***Spam* SEO:** también conocido como *spamdexing*, se refiere a la manipulación de los métodos de optimización de los motores de búsqueda para mejorar la clasificación de un sitio web y lograr que más usuarios accedan a él. Se puede clasificar en dos categorías:

    - ***Spam* de contenido:** los *spammers* llenan la página de palabras clave populares para que el sitio web aparezca más arriba en las búsquedas.
    - ***Spam* de enlaces:** se trata de comentarios que podemos encontrar en foros o redes sociales, y que contienen un enlace externo que conduce a otro sitio web. Así se consigue atraer tráfico a la página web.

- ***Spam* en redes sociales:** se trata de perfiles y cuentas falsas creadas de forma masiva con el fin de propagar un mensaje *spam*, por ejemplo, propaganda de ideas de índole política o enlaces a sitios web externos.
- ***Spam* de mensajería:** es similar al *spam* por correo electrónico pero en SMS y en plataformas de mensajería instantánea tales como WhatsApp o Telegram.
- **Estafas de soporte técnico:** suelen comenzar con la llamada telefónica de alguien haciéndose pasar por empleado de una empresa. El estafador trata de convencer al usuario de que hay un problema con su cuenta o el producto contratado, pidiendole información sensible.
- ***Spam* de malware:** este *spam* contiene enlaces o archivos que pueden comprometer la seguridad del usuario simplemente con un click o una descarga. Suele llegar a través de un mensaje de texto o un correo electrónico no deseado.

2.3. *Spam* convencional y *Phishing*

A pesar de la molestia que representa el *spam* convencional, este no suele resultar dañino, pues su objetivo es anunciar productos o servicios. Sin embargo, el *phishing* es mucho más peligroso, pues supone el envío de correos fraudulentos que se hacen pasar por mensajes de instituciones legítimas (bancos, redes sociales, instituciones públicas, etc.) [19]. Estos mensajes engañosos buscan obtener información confidencial, como contraseñas o números de tarjetas de crédito, con el fin de cometer robo de identidad o fraude financiero. Es de suma importancia que los usuarios estén capacitados para reconocer los patrones de estos intentos de *phishing* y que duden antes de enviar información sensible a terceros.

2.4. Técnicas de filtrado de *spam*

Tal y como hemos mencionado en la anterior sección, detectar el *spam* es fundamental para evitar caer en fraudes y timos online. El *spam*, especialmente en forma de *phishing*, es hoy en día la herramienta principal usada por los estafadores para engañar a los usuarios y acceder a su información confidencial.

Con el fin de proteger a los usuarios, los proveedores de servicios de correo electrónico han desarrollado durante estas décadas numerosas técnicas de detección de *spam*. En esta sección vamos a revisar el conjunto de técnicas de detección de *spam* que podemos encontrar en la literatura [20] [21]:

- **Listas negras y blancas:** también conocidas como *blacklists* y *whitelists*, este fue uno de los primeros métodos utilizados para detectar correo *spam*. Están basadas en la exclusión de mensajes que provienen de ciertos dominios, redes o servidores de Internet. Mediante este método se pueden identificar grandes cantidades de correo no deseado,

aunque bien es cierto que es fácilmente falsificable y manipulable.
Algunas de estas listas se encuentran en forma de ficheros de texto que contienen directamente remitentes o expresiones regulares de direcciones de correo electrónico.
Por el contrario, las listas blancas contienen direcciones de correo *verificadas* en las que se puede confiar. Este mecanismo, aunque simple, es muy difícil de burlar. Sin embargo, existe el inconveniente de que las empresas verificadas y exentas de estas listas pueden seguir enviando boletines de suscripción y publicidad en forma de *spam*.

- **Resúmenes:** son aplicaciones cuyo funcionamiento se basa en la creación de una representación compacta y única (resumen) de un correo electrónico. Se utilizan algoritmos de resumen, como *Nilsimsa* [22]; estos algoritmos generan una firma única para cada mensaje basándose en su contenido, de manera que incluso pequeñas modificaciones en el mensaje producen un cambio significativo en el resumen. Esto nos permite detectar variantes triviales de mensajes, como números aleatorios en el asunto o cambios mínimos en el contenido.

- **Modelos basados en contenido:** estas técnicas hacen uso del Aprendizaje Automático para determinar patrones y relaciones entre características de los mensajes clasificados como Spam y Ham. Normalmente la forma de representación de los datos es mediante un vector de características por cada correo. Las características del vector contienen una lista de palabras representativas de la legitimidad de los mensajes.
  La elección de los términos más representativos de cada mensaje se realiza mediante técnicas de selección de características. La técnica más habitual es el cálculo de la ganancia de información (IG, *Information Gain*) de cada término con respecto a los posibles valores del atributo a predecir (Spam o Ham). Se acaba tomando los términos dentro del corpus con mayor ganancia de información.
  Otra de las técnicas comúnmente empleadas en el cálculo de representatividad de un término es la frecuencia de documentos que contienen un término dado (DF, *Document Frecuency*).

- **Técnicas basadas en análisis heurístico o en reglas:** A diferencia de los enfoques basados en aprendizaje automático, el análisis heurístico no requiere un conjunto de datos de entrenamiento, sino que se basa en el conocimiento previo de los patrones típicos de *spam*. Este enfoque usa reglas o heurísticas ya creadas para evaluar una gran cantidad de patrones, que suelen ser expresiones regulares. Si el email estudiado coincide con varios de los patrones evaluados, esto va aumentando su puntuación. A su vez, si alguno de los patrones no coincide, se resta puntuación. Se establece un umbral de *spam*, y cualquier mensaje cuya puntuación lo supere se filtra como Spam; de lo contrario se considera válido.
  Estas reglas están sujetas a actualizaciones frente a la amenaza de los *spammers*, que continuamente presentan nuevos tipos de mensajes spam que podrían pasar inadvertidos ante los filtros antiguos. Un ejemplo de filtro basado en reglas es *SpamAssassin* [23].

- **Métodos basados en casos:** el filtrado basado en ejemplos es una de las técnicas de filtrado de *spam* más populares. En este enfoque, se toman todos los correos de ambas clases (Spam y Ham) y se crea una colección. Luego se llevan a cabo los pasos de preprocesamiento para transformar el email, como selección de características, agrupación de datos, etc. Los datos son clasificados en dos conjuntos de vectores. Se utiliza un algoritmo de Aprendizaje Automático (basado en instancias) para entrenar conjuntos de datos y testearlos para decidir si son Spam o Ham.

Podemos encontrar en la literatura existente numerosos métodos de filtrado de *spam* en emails. De entre las técnicas anteriomente mencionadas, algunas de ellas aplican distintos algoritmos de Aprendizaje Automático. A continuación vamos a exponer varias de las distintas técnicas de filtrado de *spam* que han sido usadas con éxito en los últimos años [20].

- **Naive Bayes**: es el algoritmo de Aprendizaje Automático más conocido en clasificación de textos. Este modelo ha adquirido gran popularidad por su capacidad de representar de forma simple y eficiente distribuciones complejas de probabilidad, además de su fácil implementación y rápida convergencia. El algoritmo se basa en el Teorema de Bayes y asume la independencia de los atributos, lo cual es una simplificación poco realista; no obstante, ha demostrado gran efectividad en numerosos estudios [24]. Se puede usar para resolver problemas de clasificación de dos o más clases, y maneja tanto datos continuos como discretos (siempre y cuando sean numéricos).

- **Máquinas de soporte vectorial** (SVM, *Support Vector Machines*): este tipo de modelos destaca por su sólida base teórica. El algoritmo se basa en la transformación de los datos existentes para encontrar un hiperplano de mayor dimensión donde maximizar la separación existente entre las clases. Cabe destacar que con SVM no es necesario realizar selección de características en la fase de Preprocesamiento, pues su capacidad de aprendizaje no se ve afectada por haber demasiados atributos.

- **Boosting de Árboles de Decisión:** se basa en tomar varios algoritmos basados en árboles de decisión y combinar las hipótesis generadas en una única hipótesis de gran precisión. Para ello, el algoritmo se ejecuta varias veces sobre distintos subconjuntos de entrenamiento. Normalmente en filtrado de *spam* se combinan algoritmos como AdaBoost y C4.5.

- **Random Forest**: es un modelo basado en una colección de árboles de decisión, entrenado cada uno a partir de un subconjunto del corpus, de forma aleatoria. Para clasificar un correo nuevo, se pasan las características del correo a cada árbol, y estos emiten cada uno un voto unitario; entonces se le asigna al correo la clase con mayor número de votos. En este algoritmo es crucial tomar un número adecuado de atributos, pues afecta de forma directamente proporcional a la correlación y a la fortaleza. Para su ajuste se suele emplear la tasa de error. Los estudios de filtrado de *spam* realizados con este clasificador han demostrado obtener un alto grado de precisión [25].

- **k-NN (*K-Nearest Neighbors*)**: es uno de los algoritmos más empleados en clasificación basada en casos o instancias. En este método, cada ejemplo de correo electrónico se representa como un punto en un espacio multidimensional, donde las dimensiones son las características del correo electrónico (por ejemplo, palabras clave, frecuencia de palabras, etc.). Luego, cuando llega un nuevo correo electrónico, se compara con los ejemplos de entrenamiento y se clasifica según la mayoría de los k correos electrónicos más cercanos en términos de similitud. Si la mayoría de los k vecinos son correos electrónicos de *spam*, entonces el nuevo correo electrónico también se clasificará como Spam. Este método es eficaz para detectar patrones sutiles en los datos y es relativamente simple de implementar.

Como hemos visto, el filtrado de emails *automático* parece ser actualmente el enfoque más exitoso para el filtrado de *spam*. Hace años, la mayor parte del correo no deseado podía detectarse simplemente analizando las direcciones de remitente y los asuntos de los emails,

mediante listas negras. Sin embargo, los *spammers* adoptaron técnicas más sofisticadas como el uso de direcciones arbitrarias y la inserción de caracteres al principio o final de la línea de asunto del mensaje.

Actualmente existen un gran número de filtros que hacen uso de una combinación de Aprendizaje Automático y conjuntos de reglas generadas a partir de la Ingeniería de Conocimiento [21]. El problema principal de utilizar reglas, es que este método no garantiza un resultado eficiente, ya que en primer lugar se debe generar un conjunto reglas (esto requerirá la ayuda de expertos en el tema) y en segundo lugar es preciso actualizar continuamente dicho conjunto. Esto puede llevar a una pérdida de tiempo y no es adecuado especialmente para usuarios inexpertos. Por otro lado, aplicando Aprendizaje Automático, no se requiere especificar ninguna regla, sino que se proporciona un conjunto de muestras de entrenamiento que son emails previamente clasificados.

3. Fundamentos matemáticos

En el contexto de la detección de correos *spam*, podemos explorar principalmente dos posibles tipos de aprendizaje: supervisado y no supervisado [26].

El **aprendizaje supervisado** consiste en entrenar un modelo a partir un conjunto de datos etiquetados, donde cada ejemplo de entrada está asociado con una etiqueta correcta (en este caso, Spam o Ham). Este enfoque es el más adecuado para el problema en cuestión, ya que permite al modelo aprender las características que distinguen los correos Spam de los Ham a partir de ejemplos previamente etiquetados. Ejemplos de técnicas de aprendizaje supervisado serían Árboles de Decisión, Regresión lineal, Redes neuronales, k-NN, SVM y métodos probabilísticos.

Por otro lado, en el **aprendizaje no supervisado** no disponemos de datos etiquetados. En su lugar, se intenta identificar patrones o estructuras subyacentes en los datos de entrada. Aunque puede ser útil para descubrir grupos de datos similares o para reducción de dimensionalidad, no es el enfoque preferido para la detección de correos electrónicos debido a la falta de orientación explícita sobre lo que constituye Spam frente a Ham. No obstante, estos métodos pueden ayudar a revelar patrones ocultos, agrupaciones o anomalías en los datos que no serían evidentes de otra manera. Ejemplos de técnicas de aprendizaje no supervisado serían Clustering (K-means, DBSCAN...), detección basada en reglas de asociación y métodos de detección de anomalías (Isolation Forest, One-Class SVM).

Existen otros dos tipos de aprendizaje: el **aprendizaje semi-supervisado** y el **aprendizaje por refuerzo** [26]. Estos enfoques se utilizan en contextos específicos donde se dispone de una cantidad limitada de datos etiquetados (semi-supervisado) o donde un agente aprende a tomar decisiones a través de recompensas (aprendizaje por refuerzo). Como hemos visto en el punto 2.4, las técnicas comúnmente utilizadas en la literatura para el filtrado de *spam* no parecen usar este tipo de aprendizaje.

En este trabajo nos vamos a centrar en el aprendizaje supervisado. En particular, dada la naturaleza binaria del problema de filtrado de correos (puede ser *spam* o no serlo), vamos a aplicar algoritmos de clasificación. La detección de correos spam es un claro ejemplo de un problema de clasificación binaria, donde cada correo electrónico se clasifica en una de dos categorías (Spam o Ham). Este problema se presta idealmente al uso de técnicas de clasificación, ya que podemos entrenar modelos utilizando conjuntos de datos previamente etiquetados para aprender a distinguir entre estas dos categorías.

La tarea de **clasificación** consiste en predecir la clase de una instancia no etiquetada, mediante un modelo. Para construir este modelo se requiere de un conjunto de instancias etiquetadas correctamente, llamado *conjunto de entrenamiento*. Podemos representar dicho conjunto por $\mathbf{E} = \{(x_i, y_i)\}_{i=1}^{n}$, formado por n instancias, donde a cada instancia x_i le corres-

ponde su etiqueta $y_i \in \{c_1, c_2, ..., c_k\}$.
Una vez que los datos de entrenamiento han sido procesados, obtenemos un modelo M, cuya función es predecir la clase $\hat{y} \in \{c_1, c_2, ..., c_k\}$ para cada nueva instancia. El conjunto de nuevas instancias es denominado *conjunto de prueba o test*. Así, para cada nuevo dato no observado x_{test}, podemos predecir su etiqueta $\hat{y} = M(x_{test})$, aplicando el modelo [3]. Podemos representar dicho conjunto por $\mathbf{P} = \{(x_j, y_j)\}_{j=1}^{m}$, donde las y_j vienen dadas por $y_j = M(x_j)$ tras la aplicación del modelo.

A lo largo de la literatura se han propuesto diversos modelos de clasificación, como árboles de decisión, clasificadores probabilísticos, máquinas de soporte vectorial y un largo etcétera, como hemos visto con más detalle en el punto 2.4. En este capítulo vamos a estudiar algunos de estos tipos de clasificadores desde sus cimientos matemáticos.

Entre los algoritmos seleccionados en este proyecto se encuentran el de Naive Bayes y las Máquinas de Soporte Vectorial (SVM, *Support Vector Machines*). Ambos algoritmos tienen sus fortalezas en el contexto de la clasificación de texto y cada uno posee características distintivas que ofrecen ventajas en términos de precisión, eficiencia y capacidad para manejar los datos de los que dispondremos, sobre otros algoritmos [3].

- Los **clasificadores probabilísticos o Bayesianos**, como Naïve Bayes, calculan la probabilidad de que una instancia pertenezca a una clase, basándose en la frecuencia de las palabras. Este clasificador resulta especialmente efectivo, debido a su simplicidad (al suponer que los atributos son independientes), permitiendo que el algoritmo sea extremadamente eficiente tanto en tiempo de entrenamiento como de predicción. A pesar de su suposición simplista de independencia, en la práctica, Naive Bayes funciona sorprendentemente bien para la clasificación de textos [27]. Además, Naive Bayes es robusto ante características irrelevantes (que suelen reducir la precisión), lo cual es una ventaja en conjuntos de atributos de alta dimensión donde no todas las características son igualmente informativas.
- Las **Máquinas de Soporte Vectorial (SVM)**, buscan el hiperplano que mejor separa las clases en el espacio de características. Es particularmente útil en espacios de alta dimensión, donde el número de características supera el número de muestras, lo cual es común en clasificación de textos [3]. Además, es conocido por su robustez y precisión, pues tiene gran capacidad para manejar la alta dimensionalidad sin caer en el sobreajuste (gracias a la selección de hiperplanos óptimos). A diferencia de Naive Bayes, SVM no asume independencia entre las características, lo que permite capturar interacciones más complejas entre ellas.

La combinación de Naive Bayes y SVM permite abordar el problema de clasificación de correos *spam* desde dos perspectivas complementarias: una probabilística y otra geométrica, maximizando así la capacidad del sistema para identificar correctamente los correos *spam*.

3.1. Clasificadores Bayesianos

Los clasificadores bayesianos son un ejemplo del enfoque probabilístico en la tarea de clasificación. Este enfoque tiene su base en el *Teorema de Bayes*, para predecir la clase que maximice

la probabilidad resultante dada una serie de eventos previos, $P(c_i|x)$. Una de las limitaciones del enfoque bayesiano es que el número de parámetros a estimar asciende a un orden de complejidad cuadrático ($O(d^2)$) [3]. Para mejorar la eficiencia computacional, los distintos clasificadores existentes en la literatura han adoptado algunas simplificaciones al problema. Por ejemplo, el clasificador *Naïve Bayes* asume que todos los atributos son independientes entre sí, lo que reduce drásticamente la cantidad de parámetros a estimar a un orden de complejidad lineal ($O(d)$) [3]. A pesar de que en la práctica, la idea de que los atributos son independientes no se suele cumplir, sorprendentemente estos clasificadores consiguen resultados muy efectivos.

Para una mejor comprensión del funcionamiento de un clasificador bayesiano, es necesario definir algunos conceptos básicos de probabilidad, que abordaremos a continuación.

3.1.1. Conceptos básicos de probabilidad

La Probabilidad es una herramienta esencial en el estudio de los fenómenos aleatorios, aquellos cuyo resultado no puede predecirse con certeza. A continuación, exploraremos los conceptos clave que nos permitirán entender mejor qué es la probabilidad y cómo se aplica en el contexto de la clasificación bayesiana.

3.1.1.1. Fenómenos y experimentos aleatorios

La Teoría de la Probabilidad estudia el comportamiento de los fenómenos o experimentos aleatorios. En primer lugar es crucial distinguir entre experimentos determinísticos y experimentos aleatorios [28].

Definición 3.1. Un *experimento determinístico* es aquel que siempre da lugar al mismo resultado bajo las mismas condiciones. Un *experimento aleatorio* es aquel cuyo resultado puede variar, a pesar de realizarse bajo idénticas condiciones.

Un ejemplo de experimento aleatorio es el de tirar un dado. Siempre estamos tirando el dado bajo las mismas condiciones; sin embargo, cada vez sale un número distinto, el cual no podemos predecir.

Una característica clave de los experimentos aleatorios es que podemos estudiar el conjunto de posibles resultados del experimento y su frecuencia, pero nunca podemos predecir un resultado particular. En el ejemplo del dado conocemos la probabilidad de que salga un número ($\frac{1}{6}$), pero no podemos predecir qué número saldrá a continuación.

3.1.1.2. Espacio muestral y Álgebra de sucesos

Definición 3.2. Denominamos *espacio muestral* al conjunto de todos los posibles resultados de un experimento aleatorio, y se denota por Ω.

En el experimento aleatorio de tirar un dado, el espacio muestral es $\Omega = \{1, 2, 3, 4, 5, 6\}$

Definición 3.3. Llamamos *suceso aleatorio* (o simplemente *suceso*) a un subconjunto del espacio muestral Ω. Decimos que se da un suceso si ocurre alguno de sus elementos como resultado del experimento aleatorio.

Por ejemplo, en el caso del dado, el suceso *obtener un número par* sería el conjunto $\{2, 4, 6\}$, que contiene los resultados 2, 4 y 6. Destacamos cuatro tipos de suceso según el número de elementos que contenga:

- **Suceso elemental**: representa cada uno de los posibles resultados del experimento aleatorio. Por tanto está formado por un único elemento del espacio muestral.
- **Suceso compuesto**: consta de dos o más elementos del espacio muestral.
- **Suceso seguro** (Ω): es aquel suceso que siempre va a ocurrir. Está formado por todos los elementos del espacio muestral.
- **Suceso imposible** ($\emptyset$): es aquel suceso que nunca va a ocurrir. No contiene ningún elemento del espacio muestral.

Esta definición de suceso como un subconjunto de Ω nos permitirá aplicar conceptos de Teoría de Conjuntos a partir de ahora. Podemos definir varias **operaciones y relaciones entre sucesos** [28]:

- **Suceso contenido en otro:** dados dos sucesos A y B de un experimento aleatorio, diremos que A está contenido en B ($A \subset B$) si siempre que ocurre el suceso A, también ocurre el suceso B.
- **Igualdad de sucesos:** dados dos sucesos A y B de un experimento aleatorio, diremos que son iguales si cada vez que ocurre A también ocurre B y viceversa.

$$A = B \iff A \subset B \text{ y } B \subset A$$

- **Suceso complementario o contrario:** dado un suceso A, se define su suceso contrario como aquel suceso que ocurre si o solo si no ocurre A. Se denota por $\overline{A}$.
- **Unión de sucesos:** dados dos sucesos A y B de un experimento aleatorio, la unión de ambos es el suceso que ocurre siempre que ocurra el A, el B o ambos a la vez. Se denota por $A \cup B$.
- **Intersección de sucesos:** dados dos sucesos A y B de un experimento aleatorio, la intersección de ambos es el suceso que ocurre cuando ocurren A y B simultáneamente. Se denota por $A \cap B$.
- **Diferencia de sucesos:** dados dos sucesos A y B de un experimento aleatorio, la diferencia de ambos es el suceso que ocurre siempre que ocurra A pero no ocurra B. Se denota por $A - B$.
- **Sucesos disjuntos o incompatibles:** dos sucesos A y B son disjuntos si no pueden ocurrir simultáneamente, es decir, si siempre que ocurre uno no se verifica el otro. Esto se cumple cuando la intersección $A \cap B = \emptyset$

Ahora, considerando estos sucesos, podemos avanzar hacia el concepto de *Álgebra de sucesos*. Este concepto nos permite realizar operaciones y establecer relaciones entre los sucesos de manera similar a como lo hacemos en el álgebra convencional.

Veremos que un *álgebra de sucesos* sobre un espacio muestral Ω es una colección de sucesos que cumple ciertas condiciones [28].

Definición 3.4. Sea Ω un conjunto arbitrario y $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ una clase no vacía de subconjuntos de Ω. La clase $\mathcal{A}$ tiene estructura de *Álgebra de sucesos* si cumple lo siguiente:

1. *Es cerrada para la operación complementario*: $\forall A \in \mathcal{A}$, se verifica que $\overline{A} \in \mathcal{A}$.
2. *Es cerrada para uniones finitas*: $\forall A, B \in \mathcal{A}$, se verifica $A \cup B \in \mathcal{A}$.

A partir de esta definición, podemos deducir de forma inmediata las siguientes propiedades:

- El espacio muestral $\Omega \in \mathcal{A}$.
- El suceso imposible $\emptyset \in \mathcal{A}$.

Hemos observado cómo los sucesos de un espacio muestral pueden ser tratados como conjuntos, permitiéndonos aplicar operaciones y relaciones entre ellos, como uniones, intersecciones y complementos sobre ellos.

Si consideramos un experimento aleatorio con espacio muestral no finito, la clase de sucesos de interés podría no ser finita, de tal forma que la estructura de álgebra dada sería insuficiente para describir la clase de sucesos $\mathcal{A} \subseteq \mathcal{P}(\Omega)$. A continuación, estudiaremos el concepto de σ-álgebra [28], que va un paso más allá al considerar también uniones *numerables* de sucesos.

Definición 3.5. Sea $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ una clase no vacía de sucesos de Ω. La clase $\mathcal{A}$ tiene estructura de *σ-álgebra de sucesos* si cumple lo siguiente:

1. *Es cerrada para la operación complementario*: $\forall A \in \mathcal{A}$, se verifica que $\overline{A} \in \mathcal{A}$.
2. *Es cerrada para uniones numerables*: $\forall A_1, A_2, A_3, ... \in \mathcal{A}$, se verifica que

$$A_1 \cup A_2 \cup A_3 \cup ... = \bigcup_{i=1}^{\infty} A_i \in \mathcal{A}.$$

Al igual que en el álgebra de sucesos, el total Ω y el vacío $\emptyset$ pertenecen a la σ-álgebra. Observamos que toda σ-álgebra es un álgebra.

3.1.1.3. Probabilidad

Teniendo presentes las definiciones de los apartados anteriores, nos vemos en las condiciones de definir el concepto de probabilidad. De forma intuitiva, la probabilidad de un suceso indica la certeza con la que puede ocurrir dicho suceso. Dicho de otra forma, la probabilidad se podría ver como la frecuencia con la que ocurre un suceso con respecto a la frecuencia con la que ocurren los demás dentro de un espacio muestral. En esta sección vamos a estudiar la definición axiomática [28], aprovechando que hemos definido anteriormente la estructura de σ-álgebra.

Definición 3.6 (Definición axiomática de Kolmogorov). Sea $(\Omega, \mathcal{A})$ el espacio medible asociado a un experimento aleatorio. Se define una probabilidad como una función de conjunto $P : \mathcal{A} \longrightarrow [0, 1]$ que verifica los siguientes axiomas:

A1 *Axioma de no negatividad*: $P(A) \geq 0, \forall A \in \mathcal{A}$.

A2 *Axioma del suceso seguro*: $P(\Omega) = 1$.

A3 *Axioma de σ-aditividad*: Dada $\{A_i\}_{i\in\mathbb{N}} \subseteq \mathcal{A}$ una familia numerable de sucesos con $A_i \cap A_j = \emptyset, \forall i \neq j$, entonces

$$P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$$

Al tomar el par $(\Omega, \mathcal{A})$, obtenemos un espacio medible, lo que significa que tenemos un marco matemático sobre el cual podemos definir y calcular probabilidades de eventos. Cada conjunto en $\mathcal{A}$ representa un posible evento en nuestro espacio muestral, y podemos asignar probabilidades a estos eventos de manera coherente y consistente.

Definición 3.7. Dados un espacio muestral Ω sobre un experimento aleatorio, su σ-álgebra de sucesos $\mathcal{A}$ y P la función de probabilidad sobre $(\Omega, \mathcal{A})$, definimos como $(\Omega, \mathcal{A}, P)$ el *espacio de probabilidades* o *espacio probabilístico* asociado al experimento.

Como consecuencia de la definición 3.6, podemos deducir varias propiedades asociadas a la probabilidad, dentro de un espacio probabilístico [28]:

1. La probabilidad del suceso imposible es nula: $P(\emptyset) = 0$.
2. La probabilidad del complementario de un suceso $A \in \mathcal{A}$ es $P(\overline{A}) = 1 - P(A)$
3. La probabilidad P es monótona no decreciente, es decir,

   $$\forall A, B \in \mathcal{A}, \text{ con } A \subset B \implies P(A) \leq P(B)$$

   Además, bajo esas mismas condiciones, $P(B - A) = P(B) - P(A)$.
4. Todo suceso $A \in \mathcal{A}$ verifica que $P(A) \leq 1$.
5. Dados dos sucesos cualesquiera $A, B \in \mathcal{A}$, se verifica que

   $$P(B - A) = P(B) - P(A \cap B)$$

6. Dados dos sucesos cualesquiera $A, B \in \mathcal{A}$, se verifica que

   $$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

7. Subaditividad finita: Dados $A, B \in \mathcal{A}$, se cumple que

   $$P(A \cup B) \leq P(A) + P(B)$$

   En general, dados $A_1, A_2, ..., A_n \in \mathcal{A}$, entonces

   $$P(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} P(A_i)$$

8. Subaditividad numerable: Dada $\{A_i\}_{i=1}^{\infty} \subset \mathcal{A}$ una familia numerable de sucesos, entonces

   $$P(\bigcup_{i=1}^{\infty} A_i) \leq \sum_{i=1}^{\infty} P(A_i)$$

9. Desigualdad de Boole: Dados $A, B \in \mathcal{A}$, entonces

$$P(A \cap B) \geq 1 - P(\overline{A}) - P(\overline{B})$$

Retomando el ejemplo del dado, supongamos que ya sabemos que ha ocurrido el evento "Ha salido un número mayor que 3". A partir de esta información, podríamos estar interesados en calcular la probabilidad de que el resultado además sea un número par. Este escenario nos hace ver la necesidad de una herramienta matemática específica que nos permita calcular la probabilidad de un suceso, a sabiendas de que otro suceso ya ha ocurrido. Este cálculo se realiza a través de la *probabilidad condicionada*.

3.1.1.4. Probabilidad condicionada

En este apartado exploraremos el concepto de *probabilidad condicionada* [28], una herramienta esencial para evaluar la probabilidad de un evento bajo la condición de que otro evento ha ocurrido. Profundizaremos en cómo este principio se extiende al *Teorema de Bayes*, permitiéndonos actualizar probabilidades a partir de nueva evidencia. Finalmente discutiremos el concepto de *independencia*, que se usa para simplificar el cálculo de probabilidades conjuntas. Estos fundamentos nos permitirán hacer inferencias más precisas y tomar decisiones en situaciones inciertas.

Definición 3.8. Sea $(\Omega, \mathcal{A}, P)$ un espacio probabilístico arbitrario y sea $A \in \mathcal{A}$ un suceso tal que $P(A) > 0$. Dado otro suceso $B \in \mathcal{A}$, definimos la *probabilidad de B condicionada a A* como

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

Observación 3.1. Observamos que de la propia definición se tiene:

$$P(A \cap B) = P(A)P(B|A), \text{ si } P(A) > 0$$

o bien

$$P(A \cap B) = P(B)P(A|B), \text{ si } P(B) > 0$$

Teorema 3.1 (de la probabilidad compuesta). *Sea $(\Omega, \mathcal{A}, P)$ un espacio probabilístico y sean $A_1, A_2, ..., A_n \in \mathcal{A}$ sucesos tal que $P\left[\bigcap_{i=1}^{n-1} A_i\right] > 0$. Entonces,*

$$P\left[\bigcap_{i=1}^{n} A_i\right] = P(A_1) \cdot P(A_2|A_1) \cdot P(A_3|A_1 \cap A_2) \cdot ... \cdot P\left[A_n| \bigcap_{i=1}^{n-1} A_i\right]$$

Teorema 3.2 (de la probabilidad total). *Sea $(\Omega, \mathcal{A}, P)$ un espacio probabilístico y sea $\{A_n\}_{n\in\mathbb{N}} \subset \mathcal{A}$ una familia de sucesos tal que $P(A_n) > 0$, $\forall n \in \mathbb{N}$. Sea $B \in \mathcal{A}$ un suceso cualquiera. Entonces,*

$$P(B) = \sum_{n=1}^{\infty} P(A_n)P(B|A_n)$$

Tras esto, nos encontramos en las condiciones para presentar el *Teorema de Bayes*. Formulado en el s.XVIII, el Teorema de Bayes ofrece un marco matemático para actualizar la probabilidad de una hipótesis a medida que se dispone de más evidencia. En esencia, el Teorema de Bayes

permite calcular la probabilidad de que ocurra un evento, basándose en el conocimiento previo de condiciones que podrían o no estar relacionadas con dicho evento.

Teorema 3.3 (Regla de Bayes o Teorema de la probabilidad inversa). *Sea $(\Omega, \mathcal{A}, P)$ un espacio probabilístico y sea $\{A_n\}_{n\in\mathbb{N}} \subset \mathcal{A}$ una familia de sucesos tal que $P(A_n) > 0, \forall n \in \mathbb{N}$. Sea $B \in \mathcal{A}$ un suceso cualquiera con $P(B) \neq 0$. Entonces,*

$$P(A_n|B) = \frac{P(B|A_n)P(A_n)}{\sum_{n\in\mathbb{N}} P(B|A_n)P(A_n)}$$

Detrás de este cálculo de probabilidades se esconde el siguiente razonamiento: supongamos que el suceso B es el resultado de aplicar un experimento, mientras que los sucesos A_n son todas las posibles causas de que ocurra el suceso B. Supongamos además que para cada causa A_n conocemos su *probabilidad a priori* $P(A_n)$ y la *verosimilitud* $P(B|A_n)$ de que el suceso B haya sido causado por A_n. Entonces la aplicación del Teorema de Bayes nos permite entender $P(A_n|B)$ como una *propiedad a posteriori* de que la verdadera causa de B haya sido A_n.

A partir de ahora, todo este apartado va a girar en torno al Teorema de Bayes, por lo que es importante comprender las implicaciones de este teorema antes de pasar a lo siguiente. Vamos a ilustrar dichas implicaciones con un ejemplo.

Ejemplo 3.1. Supongamos que tenemos un filtro de correo electrónico cuyo objetivo es clasificar los correos entrantes en Spam o Ham. Sabemos que el 20 % de los correos recibidos están clasificados como Spam. Dicho filtro utiliza palabras clave específicas para identificar el *spam*. Una de estas palabras clave es *ganar*, y se ha observado que aparece en el 30 % de los correos clasificados como Spam y sólo en el 2 % de los correos Ham. Nos planteamos la siguiente pregunta: si recibimos un correo electrónico que contiene la palabra *ganar*, ¿cuál es la probabilidad de que sea realmente *spam*?

Conocemos los siguientes datos:

- Probabilidad de recibir un correo spam: $P(Spam) = 0.2$.
- Probabilidad de recibir un correo ham: $P(Ham) = 1 - P(Spam) = 0.8$.
- Probabilidad de que la palabra *ganar* aparezca en un correo spam: $P(Ganar|Spam) = 0.3$.
- Probabilidad de que la palabra *ganar* aparezca en un correo no spam: $P(Ganar|Ham) = 0.02$.

Aplicando el Teorema de Bayes, podemos encontrar la probabilidad de que un correo sea spam sabiendo que contiene la palabra *ganar* de la siguiente forma:

$$P(Spam|Ganar) = \frac{P(Ganar|Spam)P(Spam)}{P(Ganar)}$$

Primero hay que calcular la probabilidad total de que salga la palabra *ganar*, que sabemos por el Teorema 3.2 que se calcula como

$$P(Ganar) = P(Ganar|Spam)P(Spam) + P(Ganar|Ham)P(Ham)$$

Entonces, $P(Ganar) = 0.3 \cdot 0.2 + 0.02 \cdot 0.8 = 0.076$. Por tanto, $P(Spam|Ganar) = \frac{0.3 \cdot 0.2}{0.076} = 0.789$.

En vista de los resultados, encontramos que la probabilidad de que en correo sea *spam*, dado que contiene la palabra *ganar* es del 78.9 %. Este ejemplo ilustra cómo el Teorema de Bayes permite a un filtro de correo calcular sus probabilidades basándose en la evidencia (la presencia de ciertas palabras clave), consiguiendo una clasificación más precisa. A pesar de que inicialmente solo el 20 % de los correos estaban clasificados como Spam, la aparición de la palabra *ganar* aumenta significativamente la probabilidad de que dicho correo sea Spam.

Esta idea es fundamental pues así los filtros de correo pueden aprender a partir de correos previamente etiquetados como Spam o Ham, y mejorar su precisión en el tiempo gracias a este conocimiento previo.

Ahora que hemos entendido que hay sucesos cuya probabilidad depende de que ocurran otros sucesos, vamos a estudiar la independencia de sucesos.

Definición 3.9. Sea $(\Omega, \mathcal{A}, P)$ un espacio probabilístico arbitrario y sea $A \in \mathcal{A}$ un suceso tal que $P(A) > 0$. La ocurrencia del suceso A puede alterar la probabilidad de ocurrencia de cualquier otro suceso $B \in \mathcal{A}$. Se pueden dar los casos siguientes:

1. $P(B|A) \neq P(B)$, es decir, la ocurrencia del suceso A modifica la probabilidad de ocurrencia de B. Diremos entonces que **B depende de A**.
   - Si $P(B|A) > P(B)$ se dice que el suceso A *favorece* a B.
   - Si $P(B|A) < P(B)$ se dice que el suceso A *desfavorece* a B.
2. $P(B|A) = P(B)$, es decir, la ocurrencia del suceso A no modifica la probabilidad de ocurrencia de B. Diremos entonces que **B es independiente de A**.

Observación 3.2 (**Caracterización de independencia**).

$$A \text{ y } B \text{ son independientes } \iff P(A \cap B) = P(A) \cdot P(B)$$

3.1.2. Redes probabilísticas

Las *redes probabilísticas* son un tipo de modelos probabilísticos que se caracterizan porque podemos representar de manera natural, a través de grafos, sus distribuciones de probabilidad conjunta. En estos grafos, los nodos representan las variables sobre las que definimos una distribución de probabilidad conjunta. La presencia o ausencia de enlace entre estos nodos representan la dependencia o independencia entre las variables.

Las redes probabilísticas pueden verse como representaciones de *reglas causa-efecto*, mediante las cuales podemos realizar razonamientos deductivos (conclusiones o efecto), abductivos (explicaciones o diagnóstico) e intercausales [2].

En la imagen 3.1 podemos observar un ejemplo. El razonamiento deductivo sigue la dirección de los enlaces causales entre las variables de un modelo; por ejemplo, si una persona ha contraído un resfriado, podemos concluir (con alta probabilidad) que la persona tiene fiebre y secreción nasal. Por otro lado, el razonamiento abductivo va en contra de la dirección de los enlaces causales; por ejemplo, si observamos que una persona tiene secreción nasal, tendremos evidencia para diagnosticar un resfriado o una alergia.

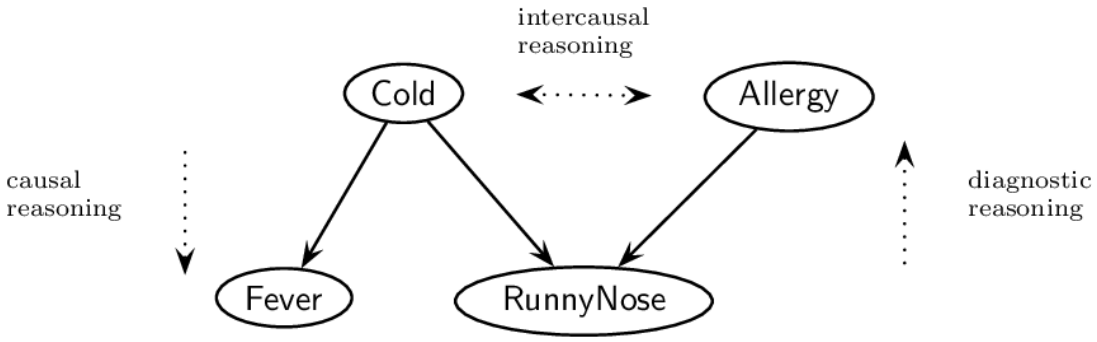


Figura 3.1.: Ejemplo de red probabilística [2]

La propiedad más característica de las redes probabilísticas es la capacidad para hacer razonamiento intercausal [2]. Esto consiste en que si obtenemos evidencia que apoya una hipótesis, la creencia en las demás hipótesis competidoras disminuye. Por ejemplo, en la figura 3.1, hay dos causas competidoras de la secreción nasal. Sin embargo, observar fiebre proporciona una evidencia muy fuerte de que el resfriado es la causa del problema, mientras que la creencia de que la alergia sea la causa disminuye sustancialmente (se descarta por la observación de la fiebre). La capacidad de las redes probabilísticas para realizar automáticamente esta inferencia intercausal es una característica clave de su poder de razonamiento.

Entre los distintos modelos basados en redes probabilísticas, dos de los ejemplos más relevantes son las redes bayesianas y las redes de Markov [29]:

- Las **redes bayesianas** utilizan grafos acíclicos dirigidos (*DAG*) para representar y para manejar la dependencia condicional entre variables. Cada nodo representa una variable aleatoria, y cada arista dirigida que conecta dos nodos indica una relación de dependencia directa. Lo que más distingue a las redes bayesianas es su capacidad para actualizar probabilidades de forma dinámica a medida que llega nueva información (inferencia bayesiana). Esto las hace muy útiles en campos como el diagnóstico médico, donde la inferencia sobre presencia de enfermedades puede mejorar con la inclusión de resultados provenientes de pruebas nuevas o síntomas adicionales observados.
- Las **redes de Markov** son estructuras que permiten modelar la transición de estados en un sistema donde el estado siguiente depende únicamente del estado actual (propiedad de Markov). Estos modelos son muy útiles en secuencias temporales donde se quiere predecir una secuencia de eventos o estados. Se usan en distintos campos como en procesamiento de señales, análisis de secuencias genéticas y PLN (Procesamiento del Lenguaje Natural).

Ambos tipos de redes ofrecen herramientas poderosas en la toma de decisiones y el análisis en un contexto de incertidumbre.

De entre estas dos redes vamos a estudiar las redes bayesianas. Mientras que las redes de Markov sirven para modelar secuencias y transiciones de estado, las redes bayesianas tienen capacidad para modelar directamente la incertidumbre y actualizar las creencias de manera incremental, lo que las convierte en una herramienta valiosa para un problema de clasificación de instancias.

Consideremos $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ un conjunto finito de variables aleatorias discretas, donde cada X_i toma valores dentro de un conjunto finito Ω_{X_i}. Una *red bayesiana* es una representación gráfica de la distribución de probabilidad conjunta. Formalmente, se trata de un par (G, Θ), donde G es un grafo dirigido acíclico (DAG) y Θ son los parámetros que indican las distribuciones de probabilidad. Los nodos del grafo G se corresponden con las variables aleatorias de $\mathbf{X}$. Entonces podemos afirmar que una red bayesiana viene dada por una componente cualitativa y otra cuantitativa [29].

- La **componente cualitativa** es el grafo dirigido acíclico $G = (\mathbf{X}, E_G)$, donde $\mathbf{X}$ es el conjunto de nodos del grafo (representando las variables del sistema), y E_G es el conjunto de arcos (representando las relaciones de dependencia directas entre dichas variables). Si dos variables están conectadas por un arco diremos que están relacionadas. En cambio, si no lo están, diremos que hay una relación de *independencia*.

  Las relaciones entre los nodos de un arco en una red bayesiana se pueden ver como una relación causa-efecto. La variable del nodo destino es dependiente del origen.

- La **componente cuantitativa** es la colección de parámetros numéricos para cada variable en $\mathbf{X}$. Estos parámetros suelen presentarse en forma de tablas de probabilidad condicional, las cuales expresan nuestras creencias sobre las relaciones entre las variables. Para cada variable X_i en el conjunto $\mathbf{X}$, existe un grupo de distribuciones condicionales, cada una correspondiente a una configuración particular de los padres de X_i en el grafo. Representamos mediante $pa_G(X) = \{Y \in \mathbf{X} | Y \longrightarrow X \in E_G\}$ al conjunto de los padres o predecesores de X. Utilizando esto, podemos calcular la distribución conjunta de todas las variables en $\mathbf{X}$. Esto se hace multiplicando las distribuciones condicionales de cada variable dada la configuración de sus padres:

$$P(x_1, ..., x_N) = \prod_{X_i \in \mathbf{X}} P(x_i | pa_G(x_i))$$

3.1.2.1. Grafos

A continuación presentamos algunas nociones básicas sobre teoría de grafos que pueden sernos de utilidad para comprender mejor las redes bayesianas ([30], [31]).

Definición 3.10. Un *grafo* G es un par (V, E), donde V es llamado *conjunto de vértices o nodos* y E *conjunto de aristas*, que representa las relaciones entre dichos vértices.

Para poder explicar cómo son las relaciones entre vértices, necesitamos definir el concepto de *grafo dirigido*.

Definición 3.11. Un *grafo dirigido* es un grafo (V, E) donde cada arista en E es un *arco dirigido*, es decir, tiene un nodo origen y un nodo destino.

En las figuras 3.2 y 3.3 podemos distinguir las diferencias entre un grafo simple y un grafo dirigido.

Para referirnos a estos nodos origen y destino que están relacionados entre sí en cada arco, vamos a hablar de nodos *padre* e *hijo*.

Definición 3.12. Un nodo X es *padre* de un nodo Y si existe un arco dirigido que va de X a Y $(X \to Y)$. Asimismo, se dice que Y es *hijo* de X. El conjunto de todos los padres de Y se denota como $pa(Y)$ y el conjunto de los hijos como $de(Y)$.

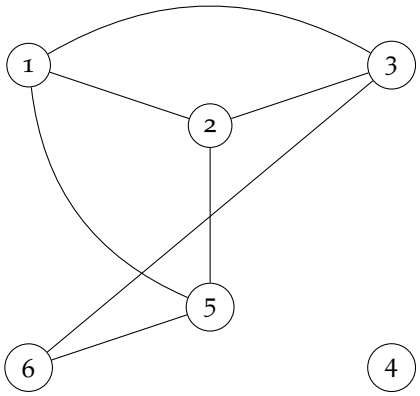


Figura 3.2.: Grafo simple

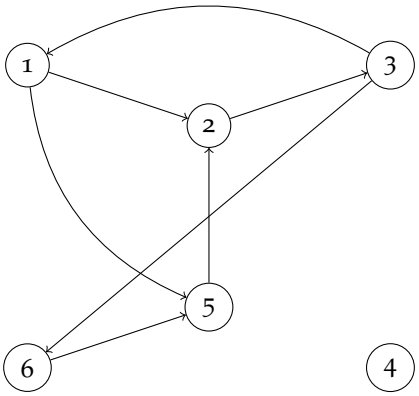


Figura 3.3.: Grafo dirigido

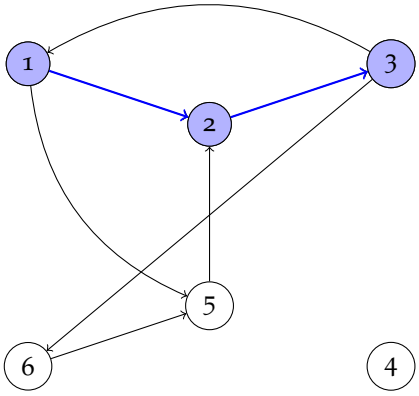


Figura 3.4.: Camino simple abierto

Definición 3.13. Un *camino* dentro de un grafo (V, E) es una sucesión de nodos $\{X_1, ..., X_N\}$, de modo que $X_i \neq X_{i+1}$, donde cada nodo está conectado con el anterior y con el siguiente, es decir, se cumple $\overrightarrow{(X_i, X_{i+1})} \in E$ o $\overrightarrow{(X_{i+1}, X_i)} \in E$, $\forall i = 1, ..., N-1$.

Definición 3.14. Un *camino dirigido* es aquel donde todas las aristas son consistentes en dirección, es decir, todas cumplen $\overrightarrow{(X_i, X_{i+1})} \in E$ o todas cumplen $\overrightarrow{(X_{i+1}, X_i)} \in E$, $\forall i = 1, ..., N-1$.

Definición 3.15. Un camino es *simple* si no pasamos más de una vez por el mismo nodo, es decir, si $X_i \neq X_j$, para $1 < i < j < N$. Además, el camino será *cerrado* si el primer y último nodo coinciden $(X_1 = X_N)$.

En la figura 3.4 podemos ver un camino entre los nodos 1 y 3, señalado en azul. Este camino es simple, pues no pasa más de una vez por el mismo nodo, y abierto, pues el primer y último nodo no coinciden. Además, es dirigido.

A partir de estos conceptos, podemos definir lo que es un *ciclo* dentro de un grafo.

Definición 3.16. Un *ciclo* es un camino cerrado simple, es decir, un camino donde $X_i \neq X_j$,

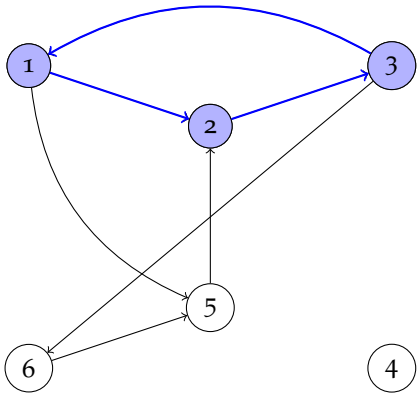


Figura 3.5.: Ciclo dirigido

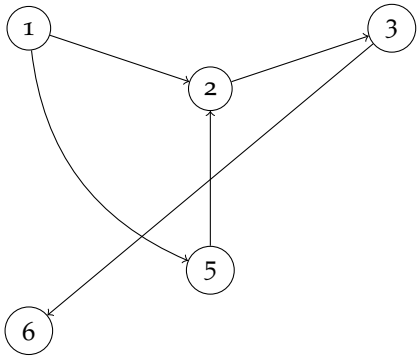


Figura 3.6.: Grafo dirigido acíclico

para $1 < i < j < N$ y donde el arco $\overrightarrow{(X_N, X_1)}$ cierra el ciclo. Si el camino además es dirigido, estaremos hablando de un *ciclo dirigido*. Si un grafo no tiene ciclos se trata de un grafo *acíclico*.

En la figura 3.5 podemos ver un ciclo dirigido.

Tras estas nociones básicas sobre Teoría de Grafos, estamos en condiciones de definir lo que es un *DAG*.

Definición 3.17. Un *grafo dirigido acíclico* (*DAG*, *Directed Acyclic Graph*), es un grafo dirigido que no contiene ningún ciclo.

El ejemplo anterior de la figura 3.5 representa un grafo que no es acíclico. El hecho de que un *DAG* no tenga ciclos, nos permitirá representar relaciones que no se repiten ni retroceden. En la figura 3.6 podemos ver un grafo dirigido sin ciclos, es decir, un *DAG*. Para ello hemos eliminado los ciclos del ejemplo anterior y hemos omitido el nodo 4 el cual no se relacionaba con ningún otro nodo.

Una ***red bayesiana*** es un modelo gráfico probabilístico que representa un conjunto de variables y sus dependencias condicionales mediante un grafo acíclico dirigido ó *DAG* [29]. Al garantizar que no haya ciclos, este grafo ayuda a prevenir paradojas o dependencias re-

cursivas infinitas en las representaciones de probabilidad.

En una red bayesiana, la distribución de probabilidad conjunta de un conjunto de variables se factoriza según la estructura del DAG. Cada nodo en el grafo representa una variable aleatoria y cada arista representa una dependencia condicional. La ausencia de ciclos asegura que se puede establecer un *orden* entre las variables, es decir, cada variable se puede expresar como condicionalmente dependiente solo de sus predecesores (sus padres) en el grafo. Esto permite que la distribución conjunta de todas las variables se escriba como el producto de las distribuciones condicionales de cada variable dado sus padres. Veremos todo esto de manera formal en el siguiente apartado.

3.1.2.2. Redes bayesianas

Definición 3.18. [32] Una *red bayesiana* es una estructura formada por:

- Un conjunto de variables proposicionales $V = \{X_1, ..., X_N\}$.
- Un conjunto E de relaciones binarias sobre las variables de V.
- Una distribución de probabilidad conjunta P definida sobre las variables de V.

Estos elementos cumplen lo siguiente:

- $\mathcal{G} = (V, E)$ es un grafo dirigido acíclico y conexo.
- $(\mathcal{G}, P)$ cumple las *hipótesis de independencia condicional*.

Decimos que un DAG (V, E) conexo junto con una distribución de probabilidad conjunta P cumple la **hipótesis de independencia condicional** si para toda variable $X_i \in V$ se tiene que el conjunto de los padres de X_i separa condicionalmente a X_i de todo subconjunto $Y \subset V$ que no contenga a X_i ni a sus hijos. En términos formales:

$$\forall X_i \in V \text{ y } \forall Y \subset V \backslash \{X_i \cup de(X_i)\} \text{ se tiene que } P(X_i|pa(X_i), Y) = P(X_i|pa(X_i))$$

Observación 3.3. Las variables en V son proposicionales en el sentido de que pueden tomar un número de estados finito. La variable $X \in V$ puede tomar los valores $\{x_1, x_2, ..., x_N\}$ Esto nos sirve para estudiar situaciones donde los estados están claramente definidos y son mutuamente excluyentes. Por ejemplo, en un diagnóstico médico, una enfermedad puede estar presente o no, y es natural representar esta situación con una variable proposicional.

En esta definición de red bayesiana, hemos partido de una distribución de probabilidad conjunta para las variables, lo cual normalmente es difícil de obtener. El siguiente resultado nos permitirá expresar esta distribución de probabilidad conjunta como producto de las distribuciones condicionadas de cada nodo dados sus padres.

Teorema 3.4 (Factorización de la probabilidad). *[32] Dada una red bayesiana, la distribución de probabilidad conjunta puede expresarse como:*

$$P(x_1, ..., x_N) = \prod_{x_i \in V} P(x_i|pa_G(x_i))$$

Demostración: *Supongamos una ordenación de las variables* $\{X_1, ..., X_N\}$ *donde los padres de cada nodo aparezcan siempre después de este. Aplicando el Teorema de la Probabilidad Total 3.2:*

$$P(x_1, ..., x_N) = \prod_{x_i \in V} P(x_i | x_{i+1}, ..., x_N)$$

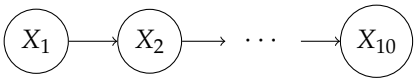
Por la forma en que hemos cogido la ordenación, el conjunto $\{X_{i+1}, ..., X_N\}$ *incluye a todos los padres de* X_i*. En consecuencia, la hipótesis de independencia condicional nos indica que:*

$$P(x_i | x_{i+1}, ..., x_N) = P(x_i | pa_G(x_i))$$

□

La ausencia de ciclos del *DAG* nos asegura que podemos establecer un *orden* entre las variables, es decir, cada variable puede ser expresada como condicionalmente dependiente solo de sus predecesores (sus padres) en el grafo. Gracias a esto, el teorema anterior nos permite escribir la distribución de probabilidad conjunta como el producto de las distribuciones condicionales de cada nodo, lo cual requiere de menos cálculos.

Ejemplo 3.2. [32] Supongamos que tenemos la siguiente red bayesiana:



Si todas las variables son binarias, necesitaríamos $2^{10} - 1$ parámetros para calcular la distribución de probabilidad conjunta. Sin embargo, si aplicamos el Teorema 3.4, las distribuciones condicionales a calcular son:

- **Distribución de X_1:** Como X_1 es la primera variable en la cadena y no tiene padres, $P(X_1 | pa_G(X_1)) = P(X_1)$. Como las variables son binarias, supongamos que X_1 toma los valores 1 o 0. Entonces bastaría con calcular $P(X_1 = 1) = p$. Esto suma 1 parámetro.
- **Distribuciones condicionales de X_2 a X_N:** Cada una de estas variables tiene un padre; entonces la probabilidad de que X_i tome un valor depende de su padre X_{i-1}. Para cada variable necesitamos especificar dos probabilidades: una cuando el padre toma el valor 1 y otra cuando toma el valor 0. Por ejemplo, para X_2 necesitamos calcular $P(X_2 = 1 | X_1 = 1)$ y $P(X_2 = 1 | X_1 = 0)$. Esto nos añade $2 \cdot 9 = 18$ parámetros.

Obtenemos un total de 19 parámetros necesarios para describir la distribución conjunta. Este ejemplo demuestra la eficiencia de las redes bayesianas para representar distribuciones conjuntas, pues hemos podido reducir muy significativamente el cálculo de probabilidades de $2^{10} - 1 = 1023$ a 19, gracias a las propiedades de independencia condicional.

3.1.3. Clasificadores basados en redes bayesianas

En el ámbito del aprendizaje automático, las redes bayesianas ofrecen un marco sólido para realizar clasificación, debido a su capacidad de modelar dependencias condicionales entre variables. Debemos tener en cuenta que dentro de una red bayesiana, cualquier variable tan solo se encuentra influenciada por su ***Markov Blanket*** (manto de Markov), que es el conjunto de sus variables padre, variables hijas y las variables que son también padre de las hijas [29]. Resulta entonces intuitivo tener en cuenta modelos clasificatorios que tengan

solo información del manto de Markov de la variable a clasificar. Por tanto se deben buscar estructuras donde todos los elementos formen parte del manto de Markov de la variable a clasificar [33].

En términos generales, un clasificador bayesiano opera bajo el principio del Teorema de Bayes, para predecir la clase que maximiza la probabilidad posterior, basándose en las evidencias observadas.

Consideremos la siguiente expresión del Teorema de Bayes orientado a un problema de clasificación, donde disponemos de n atributos $a_1, ..., a_n$. Esta expresión indica la probabilidad de que se de la clase C, dados dichos valores para los atributos.

$$P(C \mid a_1, ..., a_n) = \frac{P(a_1, ..., a_n \mid C) \cdot P(C)}{P(a_1, ..., a_n)}$$

La *hipótesis de Máximo A Posteriori (hipótesis MAP)* [34] juega un papel clave en el contexto de los clasificadores bayesianos. La hipótesis MAP busca maximizar la probabilidad posterior $P(c_i \mid \mathbf{x})$ de la clase, dado el vector de atributos $\mathbf{x}$ de una instancia.

Definición 3.19. Dada una instancia con atributos $\mathbf{x} = (x_1, \ldots, x_n)$ y la variable de clase C con k posibles categorías $\Omega = \{c_1, \ldots, c_k\}$, la hipótesis MAP se define como:

$$C_{\text{MAP}} = \arg\max_{c_i \in \Omega} P(c_i \mid x_1, \ldots, x_n)$$

Por tanto se trata de la categoría que maximiza la probabilidad posterior.

Observación 3.4. Aunque puedan parecer similares, la probabilidad condicional y la probabilidad posterior son conceptos distintos. La **probabilidad condicional** $P(A \mid B)$ describe la probabilidad de un evento A sabiendo que otro evento B ha ocurrido. Por otro lado, la **probabilidad posterior** se calcula después de observar nueva evidencia E y se utiliza para ajustar la probabilidad de una hipótesis H. Se calcula usando el Teorema de Bayes:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

Podemos ilustrarlo mejor mediante un ejemplo. Consideremos un caso médico en el que un médico evalúa la probabilidad de que un paciente tenga una enfermedad basada en la presencia de un síntoma. La probabilidad condicional sería $P(\text{Enfermedad} \mid \text{Síntoma})$. Suponiendo que se realiza una prueba médica adicional, podemos actualizar nuestra creencia sobre la enfermedad mediante la probabilidad posterior, usando el Teorema de Bayes:

$$\begin{aligned} &P(\text{Enfermedad} \mid \text{Síntoma, Resultado de prueba}) = \\ &\quad = \frac{P(\text{Resultado de prueba} \mid \text{Enfermedad}) \times P(\text{Enfermedad} \mid \text{Síntoma})}{P(\text{Resultado de prueba})} \end{aligned}$$

Esta expresión modifica la probabilidad inicial de que el paciente tenga la enfermedad, basándose en la nueva evidencia proporcionada por el resultado de la prueba.

Aplicando el Teorema de Bayes, la expresión de la hipótesis MAP queda así:

$$\begin{aligned} C_{\text{MAP}} &= \arg\max_{c_i \in \Omega} P(c_i \mid x_1, \ldots, x_n) \\ &= \arg\max_{c_i \in \Omega} \frac{P(x_1, \ldots, x_n \mid c_i) P(c_i)}{P(x_1, \ldots, x_n)} \\ &= \arg\max_{c_i \in \Omega} P(x_1, \ldots, x_n \mid c_i) P(c_i) \end{aligned}$$

donde $P(x_1, \ldots, x_n)$ es constante dada una instancia.

El problema principal de este enfoque es que hay que trabajar con la distribución conjunta y eso normalmente es inmanejable computacionalmente, sobre todo en espacios de alta dimensión y grandes conjuntos de datos.

3.1.4. Naive Bayes y tipos

El clasificador Naive Bayes es una simplificación del clasificador bayesiano general. Mientras que un clasificador bayesiano completo considera todas las posibles interacciones y dependencias entre características, Naive Bayes asume que todas las características son independientes entre sí dado el resultado de la clase. Esto simplifica notablemente los cálculos y la implementación a costa de ignorar posibles dependencias entre características. Esta simplificación hace que Naive Bayes sea menos preciso en teoría, aunque se demuestra que estos clasificadores son efectivos en la práctica, ofreciendo soluciones competentes incluso en comparación con métodos más complejos.

A continuación vamos a estudiar el enfoque general junto con los distintos clasificadores pertenecientes a la familia Naive Bayes ([29] [35] [36]).

El clasificador Naive Bayes se basa en dos supuestos [29]:

- Cada atributo $\{x_1, x_2, ..., x_n\}$ es condicionalmente independiente de los otros atributos, dada la clase C.
- Todos los atributos tienen influencia sobre la clase C.

El Teorema de Bayes 3.3 establece la siguiente relación, dada la variable de clase c_i y el vector de características $\{x_1, ..., x_n\}$:

$$P(c_i \mid x_1, \ldots, x_n) = \frac{P(c_i) P(x_1, \ldots, x_n \mid c_i)}{P(x_1, \ldots, x_n)}$$

Utilizando la suposición de independencia condicional, la cual indica que

$$P(x_j \mid c_i, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) = P(x_j \mid c_i),$$

para todo $i = 1, ..., k$, entonces

$$P(x_1, \ldots, x_n \mid c_i) = \prod_{j=1}^{n} P(x_j \mid c_i, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) = \prod_{j=1}^{n} P(x_j \mid c_i),$$

La relación anterior se simplifica a

$$P(c_i \mid x_1, \ldots, x_n) = \frac{P(c_i) \prod_{j=1}^{n} P(x_j \mid c_i)}{P(x_1, \ldots, x_n)}$$

Sabemos que el valor de la probabilidad conjunta de todas las características $P(x_1, \ldots, x_n)$ es constante, pues no cambia con respecto a diferentes clases. Por tanto podemos omitir este término en la fórmula:

$$P(c_i \mid x_1, \ldots, x_n) \propto P(c_i) \prod_{j=1}^{n} P(x_j \mid c_i)$$

Para determinar la clase más probable $\hat{c}$, elegimos la clase que maximiza este producto, y la hipótesis MAP quedaría como sigue:

$$\begin{aligned} \hat{c} = C_{\text{MAP}} &= \arg\max_{c_i \in \Omega} P(c_i \mid x_1, \ldots, x_n) \\ &= \arg\max_{c_i \in \Omega} P(c_i) \prod_{j=1}^{n} P(x_j \mid c_i) \end{aligned}$$

y entonces solo tenemos que estimar $P(c_i)$ y $P(x_i \mid c_i)$:

- $P(c_i)$ es la probabilidad a priori de la clase. Dicho de otra forma, es la frecuencia relativa de la clase c_i en el conjunto de entrenamiento. Si n es el número total de instancias y n_i el número de instancias pertenecientes a la clase c_i, entonces

$$P(c_i) = \frac{n_i}{n}$$

- $P(x_i \mid c_i)$ es cada una de las probabilidades condicionales de los atributos x_i dada la clase c_i. Veremos que según su distribución, existen distintas variantes del clasificador.

De este modo, la tarea que realiza el clasificador es buscar la clase que maximiza la función C_{MAP}.

A pesar de la simplicidad de Naive Bayes y de las restricción de que los atributos sean independientes entre sí, este clasificador consigue un alto grado de acierto, especialmente en dominios relacionados con la medicina. A la hora de realizar un diagnóstico, los médicos recogen los atributos igual que en Naive Bayes, es decir, independientes de cada clase [29].

En la figura 3.7 podemos observar dicha estructura, donde tenemos un nodo C para la clase y un nodo x_i para cada atributo. Por ejemplo, el nodo C indica si un mensaje es *spam* o *ham*, y cada x_i es un indicador de si el mensaje contiene una palabra clave (por ejemplo, *millonario*). El hecho de que el mensaje sea *spam* o no, altera la probabilidad de ocurrencia de dichas palabras, es decir, x_i depende de la clase y por tanto existe $P(x_i \mid y)$. Observamos que no hay arcos entre los x_i, pues se supone la independencia entre variables.

Los distintos clasificadores de Naive Bayes se diferencian principalmente por las suposiciones que se hacen con respecto a la distribución probabilística de $P(x_i \mid y)$. Dicha distribución puede variar dependiendo del tipo de datos y de las características.

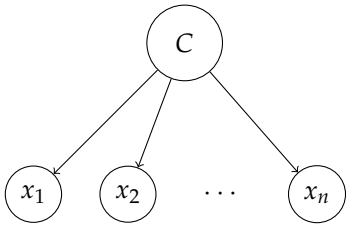


Figura 3.7.: Estructura del clasificador Naive Bayes

El clasificador **Naive Bayes simple** o binomial considera de forma binaria la probabilidad de aparición de cada término dada la clase de documento, es decir, si el término aparece o no en la clase [35]. Existen clasificadores para otras distribuciones, que pueden ser **gaussianos**, **multinomiales** o **de Bernoulli**, cada una adecuada para distintos tipos de variables de entrada (continuas, multivariantes o binarias, respectivamente).

3.1.4.1. Naive Bayes multinomial

Mientras que el clasificador Naive Bayes simple toma la aparición o no aparición de cada término en el conjunto de datos, este clasificador considera el número de apariciones de cada término para evaluar la contribución de su probabilidad condicional dada la clase del documento [35]. Este clasificador suele mejorar el desempeño del algoritmo Naive Bayes, pues se está proporcionando información adicional al clasificador para que la asignación de la clase mejore.

Este clasificador aplica Naive Bayes a datos distribuidos de forma multinomial. Esta implementación suele trabajar con vectores de conteo de palabras, y es una de las variantes de Naive Bayes más utilizadas en problemas de clasificación de textos [36] [37].

La distribución del modelo es parametrizada por vectores de la forma $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ para cada clase y, donde n es el número de características y θ_{yi} es la probabilidad de que la característica i aparezca en una muestra o instancia perteneciente a la clase y, es decir, $P(x_i \mid y)$. Si tenemos dos clases, obtendremos dos vectores θ_y, donde cada uno refleja las probabilidades de que las distintas n características sean observadas en esa clase particular.

Los parámetros θ_{yi} son estimados mediante una versión suavizada de la estimación por máxima verosimilitud (EMV), conocida como conteo de frecuencia relativa:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

En este contexto:

- $N_{yi} = \sum_{x \in T} x_i$ representa el número de veces que la característica i aparece en las muestras pertenecientes a la clase y, en el conjunto de entrenamiento T.
- $N_y = \sum_{i=1}^{n} N_{yi}$ es el conteo total de todas las características para la clase y.

- α es el parámetro de suavizado, normalmente mayor o igual a cero, el cual ayuda a manejar características que no están presentes en el entrenamiento y evita asignar probabilidades nulas. Un valor de $\alpha = 1$ es conocido como *suavizado de Laplace*, mientras que un $\alpha < 1$ como *suavizado de Lidstone*.

Si tomamos los vectores θ_y para cada clase y, obtenemos una matriz de dimensión $m \times n$, donde m es el número de clases y n el número total de características. Entonces cada fila de la matriz corresponde a una clase, y cada columna a una característica específica, donde el elemento en la fila y y columna i representa la probabilidad θ_{yi} de que la característica i aparezca en una muestra de la clase y.

En consecuencia, la matriz de parámetros Θ para un clasificador Naive Bayes Multinomial con m clases y n características se define como:

$$\Theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \cdots & \theta_{1n} \\ \theta_{21} & \theta_{22} & \cdots & \theta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{m1} & \theta_{m2} & \cdots & \theta_{mn} \end{bmatrix}$$

Ejemplo 3.3. Vamos a ver un ejemplo de aplicación del clasificador Naive Bayes Multinomial en el contexto de la filtración de correos electrónicos para identificar *spam*. En este caso, cada correo electrónico (cada instancia) se representaría como un vector de conteos de palabras. El modelo calcularía la probabilidad de que un correo pertenezca a la categoría *spam* o *ham*, basándose en la frecuencia de ciertas palabras que tienden a aparecer más en una u otra clase.

Suponemos que contamos las palabras *oferta*, *gratis* y *click* en nuestro conjunto de entrenamiento. Consideramos los siguientes conteos:

Palabra	**Spam**	**Ham**
oferta	40	5
gratis	25	3
click	30	2
Total	150	100

Utilizando un suavizado de Laplace con $\alpha = 1$, la matriz de probabilidades Θ se calcularía como sigue:

$$\Theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} = \begin{bmatrix} \theta_{\text{spam, oferta}} & \theta_{\text{spam, gratis}} & \theta_{\text{spam, click}} \\ \theta_{\text{no spam, oferta}} & \theta_{\text{no spam, gratis}} & \theta_{\text{no spam, click}} \end{bmatrix}$$

$$\Theta = \begin{bmatrix} \frac{40+1}{150+3} & \frac{25+1}{150+3} & \frac{30+1}{150+3} \\ \frac{5+1}{100+3} & \frac{3+1}{100+3} & \frac{2+1}{100+3} \end{bmatrix} = \begin{bmatrix} 0.273 & 0.173 & 0.207 \\ 0.053 & 0.035 & 0.026 \end{bmatrix}$$

donde cada fila de Θ representa una clase (primera fila para *spam*, segunda fila para *ham*), y cada columna representa una palabra (*oferta*, *gratis* y *click*).

3.1.4.2. Naive Bayes gaussiano

El clasificador Naive Bayes gaussiano [36] es una extensión de Naive Bayes para datos que siguen una distribución normal. Se utiliza cuando las características de los datos son continuas y se asume que cada característica sigue una distribución gaussiana (normal).

El modelo calcula la probabilidad de que un dato pertenezca a cierta clase, basándose en la función de densidad de probabilidad gaussiana. Se considera la media μ_y y la desviación típica σ_y de cada característica para cada clase y, para realizar la clasificación:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Los parámetros σ_y y μ_y se estiman utilizando el método de máxima verosimilitud.

Este clasificador es útil cuando las características pueden expresarse como medidas que varían de manera continua y se distribuyen normalmente, como en el caso de mediciones físicas o ciertas características financieras.

3.1.4.3. Naive Bayes de Bernoulli

El modelo Naive Bayes de Bernoulli [36] está pensado para datos que se distribuyen según distribuciones de Bernoulli multivariadas. Esto implica que puede haber múltiples características, pero cada una es binaria. Por tanto las muestras se representan como vectores de características binarias.

La regla de decisión para este clasificador se basa en la siguiente probabilidad:

$$P(x_i \mid y) = P(x_i = 1 \mid y)x_i + (1 - P(x_i = 1 \mid y))(1 - x_i)$$

Esta fórmula penaliza la no ocurrencia de una característica i que es indicativa de la clase y, a diferencia de Naives Bayes Multinomial, que ignora las características que no aparecen.

En el contexto de la clasificación de textos, para entrenar y usar este clasificador se usan vectores de ocurrencia de palabras (en lugar de conteo de palabras), que indican si una palabra se encuentra (1) o no (0) en el texto. Este clasificador resulta adecuado en conjuntos de datos con documentos más cortos, donde el conteo de frecuencias de los modelos multinomiales pueden no ser efectivos debido a la baja frecuencia de palabras, mientras que la presencia o ausencia de términos suele ser más informativa.

3.1.4.4. Clasificadores que admiten dependencias

Alternativamente, si consideramos que nuestro modelo no debería omitir las dependencias entre características, una solución sería considerar modelos TAN (*Tree Augmented Naive Bayes*) o BAN (*Bayesian Augmented Naive-Bayes*). La idea de estos modelos es relajar la restricción de que todos los atributos son condicionalmente independientes entre sí [29].

- **TAN** (*Tree Augmented Naive Bayes*): Se trata de una red bayesiana donde los atributos o variables pueden tener otro padre además de C, es decir, se pueden encontrar relaciones

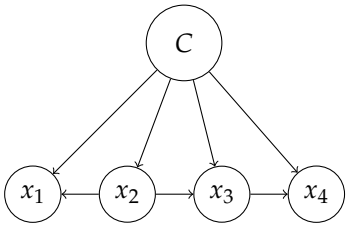


Figura 3.8.: Estructura de una red TAN

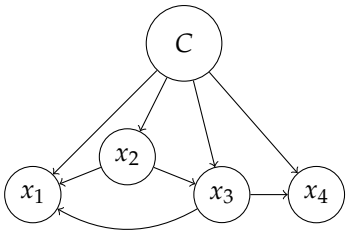


Figura 3.9.: Estructura de una red BAN

entre atributos. Por tanto la estructura de la red ya no es trivial, sino que se tienen que obtener las relaciones entre atributos. La red pasa a tener estructura de árbol. Estas estructuras obtienen mejores resultados que Naive Bayes, al mismo tiempo que no aumentan demasiado la complejidad computacional, y conservan la robustez del predecesor. Un ejemplo de la estructura de una red TAN es la mostrada en la figura 3.8.

- **BAN** (*Bayesian Augmented Naive-Bayes*): Partiendo de la estructura del TAN, ahora relajamos completamente la restricción de que los atributos sean independientes entre sí. Cada atributo podrá tener varios padres además de la clase C. De esta forma, la estructura ya no tiene por qué ser un árbol. En la figura 3.9 podemos ver un ejemplo de red BAN.

- **Semi Naive Bayes**: Este otro enfoque [29] consiste en tomar los atributos correlacionados entre sí y reunirlos en un atributo compuesto. Así se consigue un clasificador Naive Bayes usual, pues los nuevos atributos compuestos se consideran independientes entre ellos dada la clase.

  En la figura 3.10 podemos ver un ejemplo. Supongamos que las variables x_1, x_2 y x_3 predicen la clase C. Supongamos que las variables x_1 y x_3 son condicionalmente dependientes dada C. Entonces podríamos agruparlas en una variable compuesta, donde las variables $\{x_1, x_3\}$ y x_2 son independientes.

3.1.4.5. Dependencias y su impacto en Naive Bayes

La suposición fundamental de Naive Bayes es que los atributos son condicionalmente independientes dados los valores de clase. Matemáticamente, esto se expresa en el Teorema 3.3.

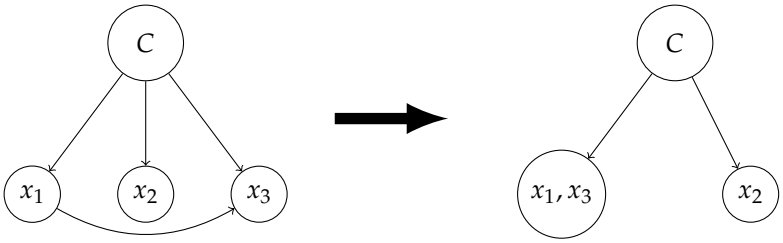


Figura 3.10.: Ejemplo de clasificador Semi Naive Bayes

A pesar de esta suposición, la realidad es que los atributos a menudo están correlacionados. Sin embargo, el impacto de estas dependencias puede ser mitigado o incluso cancelado bajo ciertas condiciones. Según [38], esto se debe a que las interacciones entre dependencias pueden trabajar en direcciones opuestas, neutralizando el efecto neto sobre la probabilidad condicional de clase.

Consideremos dos atributos x_1 y x_2 que son dependientes entre sí. Aunque esta dependencia viola la suposición de independencia de Naive Bayes, su efecto en la decisión final de clasificación puede ser insignificante si estas dependencias se distribuyen de manera uniforme a través de las clases o si se cancelan mutuamente.

A partir de ahora vamos a tratar con una red bayesiana aumentada o BAN, donde además de compartir todos los nodos un mismo nodo padre, pueden también estar relacionados entre sí de forma libre. Consideraremos dos clases, denotadas como {+, -}.

La dependencia local de un nodo se refiere a cómo los nodos padres de un nodo específico influyen en este último dentro de cada clase. Para cuantificar esta influencia, se utiliza la razón entre la probabilidad condicional del nodo dado sus padres y la probabilidad condicional del nodo sin considerar a sus padres. Matemáticamente, se define de la siguiente manera:

Definición 3.20. Dado un nodo X en una BAN G, la *derivada de dependencia local* de X en las clases {+, -} se define como sigue:

$$dd_G^+(x|pa(x)) = \frac{p(x|pa(x),+)}{p(x|+)}$$

$$dd_G^-(x|pa(x)) = \frac{p(x|pa(x),-)}{p(x|-)}$$

La medida $dd_G^+(x|pa(x))$ refleja la fuerza de la dependencia local del nodo X para la clase positiva (+), indicando cuánto la presencia de los padres afecta la probabilidad de X en dicha clase. Análogamente, $dd_G^-(x|pa(x))$ actúa de la misma forma para la clase negativa (-).

Observación 3.5. De la anterior definición se extraen los siguientes resultados:

- Cuando X no tiene padres, entonces:

$$dd_G^+(x|pa(x)) = dd_G^-(x|pa(x)) = 1.$$

- Cuando $dd_G^+(x|pa(x)) \geq 1$, la dependencia local de X apoya la clasificación de la clase positiva (pues el numerador es mayor que el denominador). De lo contrario, apoya la clasificación de la clase negativa. Oucrre de forma análoga cuando $dd_G^-(x|pa(x)) \geq 1$.

Definición 3.21. Para un nodo X en una BAN G, la *razón de la derivada de dependencia local* en el nodo X se define como sigue:

$$ddr_G(x) = \frac{dd_G^+(x|pa(x))}{dd_G^-(x|pa(x))}$$

Observación 3.6. De acuerdo con la definición anterior, $ddr_G(x)$ cuantifica la influencia de la dependencia local de X en la clasificación. Además, extraemos los siguientes resultados:

1. Si X no tiene padres, entonces $ddr_G(x) = 1$.
2. Si $dd_G^+(x|pa(x)) = dd_G^-(x|pa(x))$, entonces $ddr_G(x) = 1$. Esto significa que la dependencia local de x se distribuye uniformemente en la clase + y en la clase -. Por lo tanto, la dependencia no afecta la clasificación, sin importar cuán fuerte sea.
3. Si $ddr_G(x) > 1$, la dependencia local de X en la clase + es más fuerte que en la clase -. Si $ddr_G(x) < 1$, significa lo contrario.

El resultado más relevante que extraemos es que cuando el valor de $ddr_G(x)$ es cercano a 1, entonces la dependencia entre x_1 y x_2 no afecta significativamente a la clasificación. Por ejemplo, si en una red bayesiana x_1 es padre de x_2, entonces $ddr_G(x) = 1$ indica que la dependencia de x_1 con sus padres es similar para ambas clases {+,-}, sugiriendo que esta dependencia no juega un papel crucial a la hora de clasificar en la clase positiva o la negativa.

Definición 3.22. Dada una red bayesiana aumentada (BAN), definimos el *factor de distribución de dependencia global* para un ejemplo E como:

$$DFG(E) = \prod_{i=1}^{n} ddrG(x_i)$$

donde E es un ejemplo con atributos $x_1, x_2, \ldots, x_n$.

Este factor mide cómo las dependencias locales entre atributos afectan la clasificación del ejemplo E. Si $DFG(E) \approx 1$, entonces las dependencias se consideran balanceadas o que se cancelan entre sí, permitiendo que el clasificador mantenga un buen rendimiento.

Cuando $DFG(E) = 1$, las dependencias modeladas en la red bayesiana G no tienen ninguna influencia en la clasificación. En otras palabras, la clasificación de G es exactamente igual a la de su correspondiente clasificador Naive Bayes. Existen tres casos en los que $DFG(E)$ puede ser igual a uno:

1. **No existe dependencia entre atributos:** Esto implica que cada atributo es independiente de los demás, lo que valida la suposición de independencia de Naive Bayes.
2. **Distribución local equitativa entre clases:** Para cada atributo X en G, $ddr_G(x) = 1$. Esto significa que la distribución de la dependencia local de cada nodo es uniforme entre las clases, neutralizando cualquier influencia en la clasificación debido a dependencias.

3. **Cancelación de influencias en la clasificación:** La influencia de algunas dependencias locales que apoyan la clasificación de E en la clase positiva es contrarrestada por la influencia de otras dependencias locales que apoyan la clasificación de E en la clase negativa. Esto da lugar a un balance que neutraliza el efecto neto de las dependencias en la clasificación final.

Hemos probado que bajo ciertas condiciones, un modelo BAN puede comportarse de manera equivalente que un modelo de Naive Bayes simple, a pesar de las dependencias entre atributos.

Esto demuestra que a pesar de que Naive Bayes opera teóricamente bajo suposiciones de independencia, su robustez se puede mantener aunque estas suposiciones no se cumplan. Las dependencias entre atributos pueden no afectar de forma negativa al rendimiento de Naive Bayes siempre y cuando estas dependencias se cancelen entre sí, o dicho de otra forma, mientras se distribuyan uniformemente a través de las clases. Esta propiedad convierte a Naive Bayes en un clasificador resistente a la presencia de dependencias entre atributos.

3.2. Máquinas de Soporte Vectorial

En esta sección vamos a describir las **Máquinas de Soporte Vectorial** (*Support Vector Machines*, SVM) [3]. Se trata de un método de clasificación basado en discriminantes lineales de margen máximo. El objetivo es encontrar el hiperplano lineal más óptimo, es decir, aquel que maximice la distancia entre clases. Veremos además que para aquellos casos donde no se puede encontrar dicho hiperplano lineal, se puede aplicar el *truco del kernel*, por el cual se transforman los datos a un espacio de dimensión superior donde sí pueden ser separados linealmente.

3.2.1. Hiperplanos

Sea $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^{n}$ un conjunto de datos, formado por n puntos, en un espacio d-dimensional. Además, asumamos que solo hay dos etiquetas de clase, es decir, $y_i \in \{+1, -1\}$, denotando las clases positiva y negativa.

Definición 3.23. [3] Definimos un ***hiperplano*** en un espacio d-dimensional, como aquel formado por los puntos $\mathbf{x} \in \mathbb{R}^d$ que cumplen la siguiente ecuación:

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\ &= \langle \mathbf{w}, \mathbf{x} \rangle^{1} = 0 \end{aligned}$$

donde $\mathbf{w}$ es un vector de pesos d-dimensional, y b es un escalar, denominado *sesgo*. Entonces

$$h(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = 0$$

Observación 3.7. Por tanto, podemos ver el hiperplano como el conjunto de todos los puntos tal que $\mathbf{w}^T\mathbf{x} = -b$. Para ver el papel que juega b, suponiendo que $w_1 \neq 0$ y que $x_i = 0 \forall i > 1$, obtenemos el punto donde el hiperplano interseca el primer eje:

$$w_1x_1 = -b \quad \Rightarrow \quad x_1 = -\frac{b}{w_1}$$

Entonces el punto $(-\frac{b}{w_1}, 0, ..., 0)$ se encuentra en el hiperplano.

De igual forma, podemos hallar los puntos de corte del hiperplano con cada uno de los ejes, dado por $-\frac{b}{w_i}$ con $w_i \neq 0$.

A continuación tiene sentido preguntarse qué ocurre con los puntos que no cumplen la ecuación. Estos hacen que la expresión $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ sea mayor o menor que cero. En dicho caso, el punto estará a un lado u otro del hiperplano. Por tanto, un hiperplano $h(x)$ divide el espacio original d-dimensional en dos partes.

Definición 3.24. Un conjunto de datos es ***linealmente separable*** si podemos encontrar un hiperplano separador $h(x)$, donde para todos los puntos etiquetados con $y_i = -1$, tenemos $h(x_i) < 0$, y para todos los puntos etiquetados con $y_i = +1$, tenemos $h(x_i) > 0$.

Entonces el hiperplano $h(x)$ puede verse bien como un clasificador lineal o como un ***discriminante lineal***, el cual predice la clase y para cualquier punto x, de acuerdo a la

[1] Ver definición B.1 en Glosario

siguiente regla:

$$y = \begin{cases} +1 & \text{si } h(x) > 0 \\ -1 & \text{si } h(x) < 0 \end{cases}$$

Observación 3.8. Supongamos que $\mathbf{a_1}$ y $\mathbf{a_2}$ son dos puntos arbitrarios que yacen sobre el hiperplano. Entonces cumplen

$$h(\mathbf{a_1}) = \mathbf{w}^T\mathbf{a_1} + b = 0$$

$$h(\mathbf{a_2}) = \mathbf{w}^T\mathbf{a_2} + b = 0$$

Restando ambas expresiones obtenemos

$$\mathbf{w}^T(\mathbf{a_1} - \mathbf{a_2}) = 0$$

Esto significa que el vector de pesos $\mathbf{w}$ es ortogonal al hiperplano, ya que es ortogonal a cualquier vector arbitrario $(\mathbf{a_1} - \mathbf{a_2})$ en el hiperplano. En otras palabras, el vector de peso $\mathbf{w}$ es el vector normal al hiperplano, el cual fija su orientación.

A continuación veremos que gracias a $\mathbf{w}$ podemos definir la distancia de un punto cualquiera al hiperplano.

3.2.2. Distancia de un punto a un hiperplano

Consideremos un punto $\mathbf{x} \in \mathbb{R}^d$ fuera del hiperplano. Sea $\mathbf{x}_p$ la proyección ortogonal de $\mathbf{x}$ sobre el hiperplano, y sea $\mathbf{r} = \mathbf{x} - \mathbf{x}_p$, entonces podemos escribir $\mathbf{x}$ como

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_p + \mathbf{r} \\ &= \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|} \end{aligned}$$

donde $\mathbf{r}$ es la distancia dirigida del punto $\mathbf{x}$ a $\mathbf{x}_p$, es decir, $\mathbf{r}$ da el desplazamiento de $\mathbf{x}$ a $\mathbf{x}_p$ en términos del vector unitario $\mathbf{w}$. El desplazamiento $\mathbf{r}$ es positivo si $\mathbf{r}$ está en la misma dirección que $\mathbf{w}$, y $\mathbf{r}$ es negativo si $\mathbf{r}$ está en una dirección opuesta a $\mathbf{w}$. Por tanto podemos expresarlo como $\mathbf{r} = r\frac{\mathbf{w}}{\|\mathbf{w}\|}$, tal y como se observa en la figura 3.11.

Aplicando la ecuación del hiperplano, obtenemos

$$\begin{aligned} h(\mathbf{x}) &= h(\mathbf{x}_p + \mathbf{r}) \\ &= \mathbf{w}^T(\mathbf{x}_p + \mathbf{r}) + b \\ &= \mathbf{w}^T(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + b \\ &= \underbrace{\mathbf{w}^T\mathbf{x}_p + b}_{h(\mathbf{x}_p)} + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} \\ &= \underbrace{h(\mathbf{x}_p)}_{0} + r\|\mathbf{w}\| \\ &= r\|\mathbf{w}\| \end{aligned}$$

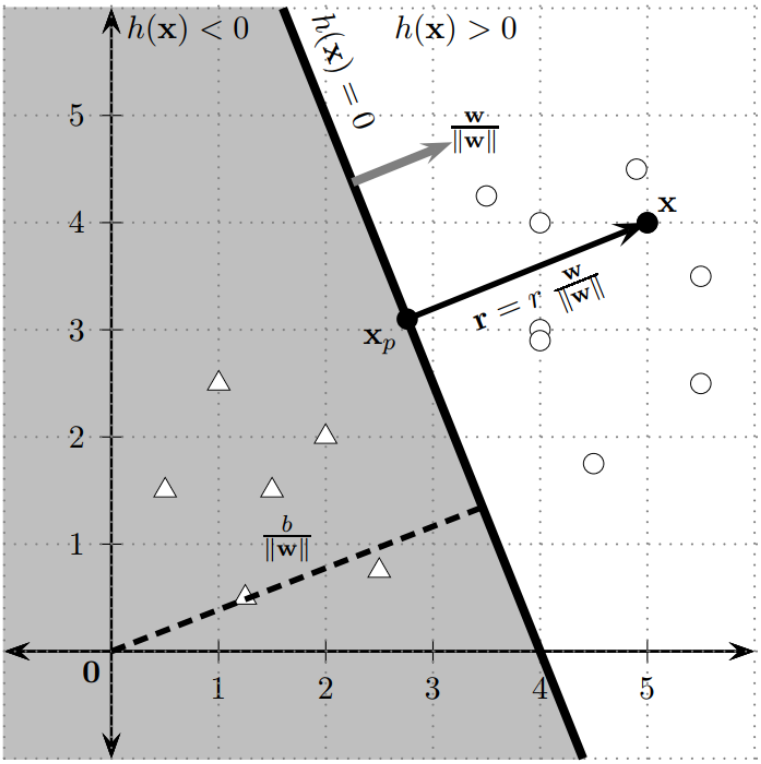


Figura 3.11.: Distancia de un punto a un hiperplano en un espacio de dimensión 2 [3]

donde el término $\mathbf{w}^T\mathbf{w}$ representa el producto escalar del vector $\mathbf{w}$ consigo mismo, lo que es igual que la norma al cuadrado $\|\mathbf{w}\|^2$. El último paso se sigue del hecho de que $h(\mathbf{x}_p) = 0$, ya que $\mathbf{x}_p$ es un punto del hiperplano. Gracias al resultado anterior, obtenemos una expresión para la distancia directa del punto $\mathbf{x}$ al hiperplano:

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

Observación 3.9. Para obtener la distancia real, necesitamos imponer que sea positiva. Para ello, podemos multiplicar r por la etiqueta de la clase y para el punto $\mathbf{x}$. Por tanto, la distancia del punto $\mathbf{x}$ al hiperplano se expresa como:

$$\delta = yr = \frac{yh(\mathbf{x})}{\|\mathbf{w}\|}$$

donde $y \in \{+1, -1\}$.

Observación 3.10. En particular, para el origen $\mathbf{x} = 0$, la distancia dirigida es

$$\mathbf{r} = \frac{h(0)}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T 0 + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|}$$

tal y como se observa en la figura 3.11.

3.2.3. Hiperplano óptimo

En la sección anterior hemos estudiado que dado un conjunto de datos $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^{n}$, donde $y_i \in \{+1, -1\}$, y dado un hiperplano de separación $h(\mathbf{x}) = 0$, podemos definir la distancia de cualquier punto $\mathbf{x}_i$ al hiperplano como

$$\delta_i = \frac{y_i h(\mathbf{x}_i)}{\|\mathbf{w}\|}$$

A continuación estudiaremos los conceptos de margen y vector soporte para definir el hiperplano óptimo, es decir, el que maximiza la distancia al hiperplano de los puntos más cercanos de cada clase.

Definición 3.25. Definimos el ***margen*** del clasificador como la distancia mínima de un punto al hiperplano, dado por

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i h(\mathbf{x}_i)}{\|\mathbf{w}\|} \right\}$$

Observación 3.11. Nótese que $\delta^* \neq 0$, ya que $h(\mathbf{x})$ es un hiperplano separador, y por tanto $h(\mathbf{x}_i) \neq 0$ para cualquier punto fuera del hiperplano.

Definición 3.26. Llamamos ***vectores de soporte*** a aquellos puntos que se encuentran a distancia δ^* de hiperplano. En otras palabras, un vector de soporte $\mathbf{x}^*$ es un punto que se encuentra en el margen del clasificador, y que satisface

$$\delta^* = \frac{y^* h(\mathbf{x}^*)}{\|\mathbf{w}\|}$$

donde y^* es la etiqueta para la instancia $\mathbf{x}^*$.

Observación 3.12. El numerador $y^* h(\mathbf{x}^*) = y^*(\mathbf{w}^T \mathbf{x}^* + b)$ indica la distancia absoluta del vector de soporte $\mathbf{x}^*$ al hiperplano, mientras que el denominador $\|\mathbf{w}\|$ la convierte en una distancia relativa en términos de $\mathbf{w}$.

Considerando la ecuación del hiperplano (3.23), si multiplicamos ambos lados por un escalar $s \in \mathcal{R}$, obtenemos un hiperplano equivalente

$$sh(\mathbf{x}) = s(\mathbf{w}^T \mathbf{x} + b) = (s\mathbf{w})^T \mathbf{x} + (sb) = 0$$

Gracias a esto podemos definir infinitos hiperplanos que separan el conjunto de datos, pero solo queremos uno, por lo que definimos el hiperplano *canónico* como sigue.

Definición 3.27. Definimos el hiperplano *canónico* como aquel cuya distancia absoluta a los vectores de soporte es 1. Es decir,

$$sy^* h(\mathbf{x}^*) = 1$$

lo que implica

$$s = \frac{1}{y^* h(\mathbf{x}^*)}$$

De aquí en adelante, asumiremos que cualquier hiperplano separador es canónico. Es decir, ya ha sido escalado adecuadamente para que $y^* h(\mathbf{x}^*) = 1$ para un vector de soporte $\mathbf{x}^*$, y el margen del clasificador se expresa como

$$\delta^* = \frac{y^* h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

tal y como podemos apreciar en la figura 3.12.

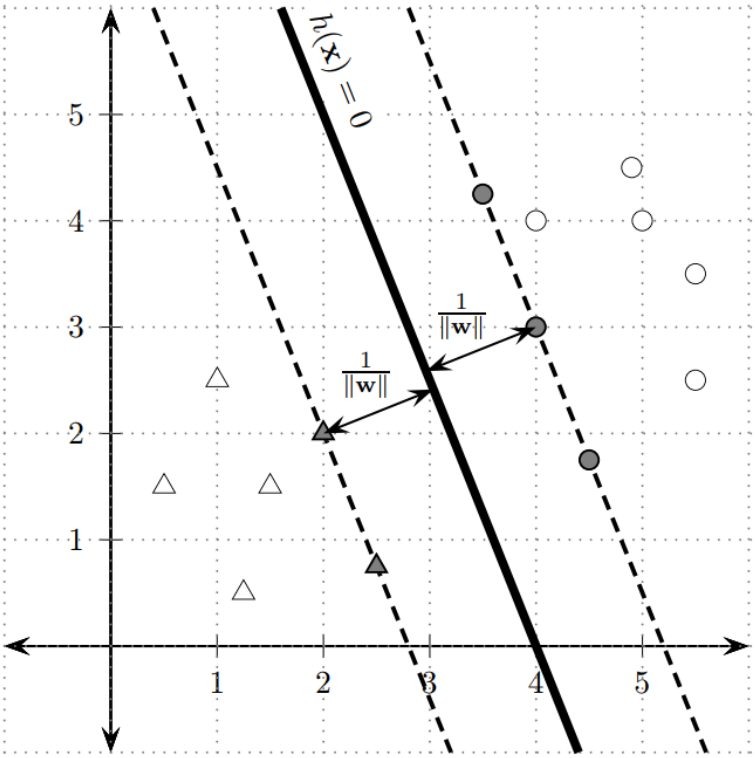


Figura 3.12.: Margen de separación en un hiperplano canónico [3]

Observamos que para cualquier punto que no sea vector soporte, su distancia al hiperplano es $\delta_i > \frac{1}{\|\mathbf{w}\|}$. Entonces para los puntos del conjunto de datos $\mathbf{D}$ obtenemos el siguiente conjunto de desigualdades:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall \mathbf{x}_i \in \mathbf{D}.$$

3.2.4. Problema lineal

A continuación veremos dos enfoques distintos para el caso lineal: SVM de margen duro (*hard margin SVM*) y SVM de margen blando (*soft margin SVM*). Cada una aporta la solución más óptima según el conjunto de datos $\mathbf{D}$ sea linealmente separable o no.

3.2.4.1. SVM de margen duro

Supongamos un conjunto de datos $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^n$, con $x_i \in \mathbb{R}^d$ e $y_i \in \{+1, -1\}$. Supongamos además que los puntos son **linealmente separables**, es decir, según la definición 3.24

existe un hiperplano separador $h(x)$ que clasifica correctamente todos los puntos. El problema es que, tal y como vimos en el punto 3.2.3, existe un número infinito de hiperplanos separadores. ¿Cuál debemos elegir entonces?

La idea fundamental detrás de las SVM es seleccionar el hiperplano canónico [3], dado por el vector de pesos $\mathbf{w}$ y el sesgo b, que ofrece el mayor margen entre todos los posibles hiperplanos separadores $h(\mathbf{x}) \equiv \mathbf{w}^T\mathbf{x} + b = 0$. Si δ_h^* representa el margen para el hiperplano $h(\mathbf{x}) = 0$, entonces el objetivo es encontrar el hiperplano óptimo h^*:

$$h^* = \arg\max_h \delta_h^* = \arg\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|}$$

El objetivo del clasificador SVM es entonces encontrar el hiperplano que maximiza el margen $\frac{1}{\|\mathbf{w}\|}$, lo cual es equivalente a minimizar $\|\mathbf{w}\|$. Por tanto, con el fin de encontrar el hiperplano óptimo, nos enfrentamos a un problema de optimización que se puede formular de la siguiente manera:

$$\begin{aligned} \textbf{Función objetivo:} \quad & \min_{\mathbf{w},b} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 \right\}, \\ \textbf{Restricciones lineales:} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall \mathbf{x}_i \in \mathbf{D}. \end{aligned}$$

Observación 3.13. La razón detrás de minimizar $\frac{\|\mathbf{w}\|^2}{2}$ en lugar de $\|\mathbf{w}\|$ se debe a cuestiones técnicas. En primer lugar, $\|\mathbf{w}\|^2$ es diferenciable en todo su dominio mientras que $\|\mathbf{w}\|$ no lo es en 0, lo cual puede complicar la optimización.

Por otro lado, $\|\mathbf{w}\|^2$ es estrictamente convexo, que es una propiedad deseable ya que garantiza que hay un único mínimo, que es global. Esto es crucial en aprendizaje automático, donde queremos asegurarnos de que el modelo no se atasque en mínimos locales durante la fase de entrenamiento.

Aunque menos relevante, otro detalle es que la presencia del término $\frac{1}{2}$ es conveniente para simplificar derivadas, ya que cuando tomamos la derivada de $\frac{1}{2}\|\mathbf{w}\|^2$, el factor $\frac{1}{2}$ anula el 2 que aparece al derivar $\|\mathbf{w}\|^2$. Esto hace que los cálculos sean más rápidos y tengan menos errores numéricos.

La formulación anterior garantiza que todos los puntos de datos estén al menos a una distancia $1/\|\mathbf{w}\|$ del hiperplano, con los vectores de soporte exactamente a esta distancia, y todos los demás puntos correctamente clasificados más allá de este margen.

El problema de optimización anterior se denomina problema *primal*. Podemos resolver este problema primal con algoritmos de optimización estándar [3]. Sin embargo, es más común resolver el problema *dual*. Según la teoría de optimización dual [39], un problema de optimización primal tiene una forma dual asociada, siempre y cuando la función objetivo y las restricciones sean estrictamente convexas. Para expresar la forma dual, necesitamos multiplicadores de Lagrange.

Definición 3.28. Dado un problema de optimización con variable de decisión $\mathbf{w}$, función objetivo $h(\mathbf{w})$ y las restricciones $g_i(\mathbf{w}) = 0$, $i = 1, ..., n$, definimos la **función de Lagrange**

como

$$L(\mathbf{w}, \boldsymbol{\alpha}) = h(\mathbf{w}) + \sum_{i=1}^{n} \alpha_i g_i(\mathbf{w})$$

donde $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$ con $\alpha_i \geq 0$, $\forall i = 1, ..., n$. Los términos α_i son conocidos como los **multiplicadores de Lagrange**.

En nuestro problema de optimización, podemos observar que la función objetivo es estrictamente convexa por tratarse del cuadrado de una norma, y las restricciones por ser funciones lineales. Por tanto estamos en las condiciones de resolver el **problema dual**. Los resultados y los detalles necesarios para llegar al problema dual se pueden consultar en [3], [39] y [40].

$$\begin{aligned} \textbf{Función objetivo:} \quad & \max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right\}, \\ \textbf{Restricciones lineales:} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \\ & \alpha_i \geq 0, \quad \forall i = 1, ..., n. \end{aligned}$$

Los multiplicadores de Lagrange desempeñan un papel crucial en este problema, ya que identifican a los vectores de soporte. Los vectores de soporte son aquellos puntos para los cuales $\alpha_i > 0$ y se encuentran en el margen o violan el margen. Los puntos tal que $\alpha_i = 0$ se encuentran fuera de los márgenes.

La solución del poblema dual, $\boldsymbol{\alpha}$, nos permitirá obtener la solución del problema primal, es decir, los parámetros $\mathbf{w}$ y b que maximizan el margen. Según [3] y [26], estos vienen dados por:

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \\ b_i &= y_i - \mathbf{w}^T \mathbf{x}_i \\ &\Downarrow \\ b = avg_{\alpha_i > 0}\{b_i\} &= \frac{1}{n_s} \sum_{\substack{i=1 \\ \alpha_i > 0}}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i), \end{aligned}$$

donde n_s es el número total de vectores de soporte.

Por tanto, $\mathbf{w}$ se obtiene como combinación lineal de los vectores de soporte, con los α_i representando los pesos [3]. Observamos que los puntos que no son vectores de soporte, no aportan al cálculo de $\mathbf{w}$ pues en estos casos $\alpha_i = 0$.

El término b, demoninado sesgo o *bias*, representa el desplazamiento desde el origen del hiperplano de decisión, dentro del espacio de características.

Sustituyendo la solución óptima $\mathbf{w}$, nos queda la siguiente expresión para el hiperplano óptimo:

$$h(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b, \quad \mathbf{x} \in \mathbb{R}^d.$$

Para clasificar un nuevo punto $\mathbf{x}^*$, basta con evaluarlo en el expresión hiperplano óptimo, de tal forma que:

- Si $h(\mathbf{x}^0) \geq 0$, entonces $\mathbf{x}^0$ tiene la etiqueta $y^0 = +1$.
- Si $h(\mathbf{x}^0) < 0$, entonces $\mathbf{x}^0$ tiene la etiqueta $y^0 = -1$.

3.2.4.2. SVM de margen blando

En ocasiones, es frecuente que el conjunto de datos $\mathbf{D}$ no sea perfectamente separable mediante una función lineal, como habíamos supuesto hasta ahora. Esto se traduce en que si usamos un hiperplano lineal, habrá puntos que se han clasificado incorrectamente. Luego en cada lado del hiperplano podríamos encontrar al mismo tiempo datos con etiqueta $y_i = +1$ y con etiqueta $y_i = -1$.

En esta sección vamos a adaptar el problema de optimización anterior para el **caso lineal no separable**, que denominamos SVM de margen blando (*soft margin SVM*). En la figura 3.13 podemos visualizar la diferencia entre SVM de margen duro y SVM de margen blando.

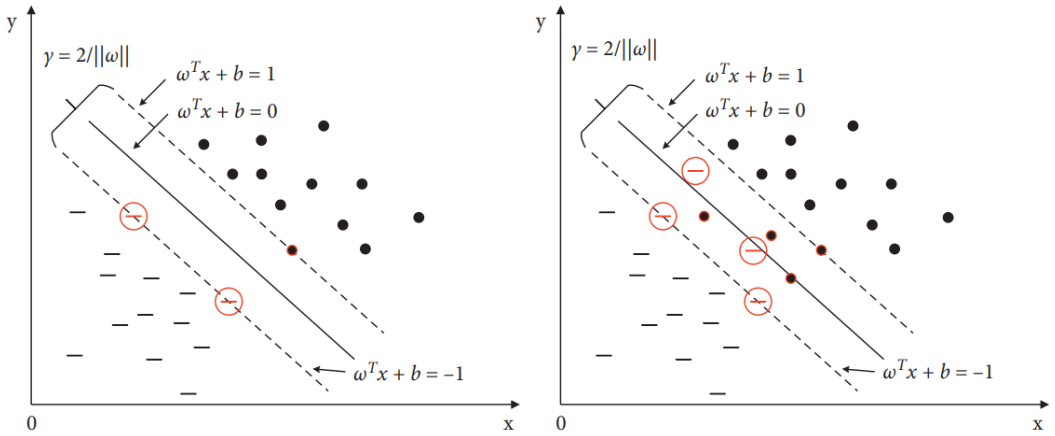


Figura 3.13.: Comparación entre SVM de margen duro y SVM de margen blando [4]

Recordemos que cuando el conjunto de datos era linealmente separable, podíamos encontrar un hiperplano donde se cumpliera $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \forall \mathbf{x}_i \in \mathbf{D}$. Ahora, para el caso de datos que no son linealmente separables, introducimos las *variables de holgura* $\xi_i \geq 0$ para cada punto $\mathbf{x}_i \in \mathbf{D}$. Estas variales miden la desviación del punto $\mathbf{x}_i$ con respecto a la zona a la que deberían pertenecer. La nueva restricción que deben cumplir los puntos es:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \in \mathbf{D}$$

Observación 3.14. Dependiendo del valor de la holgura, podemos tratar con 3 tipos de puntos:

- Si $\xi_i = 0$, entonces $\mathbf{x}_i$ está como mínimo a una distancia $\frac{1}{\|\mathbf{w}\|}$.
- Si $0 < \xi_i < 1$, entonces el punto se encuentra dentro del margen pero aún está correctamente clasificado.

- Si $\xi_i \geq 1$, entonces el punto está mal clasificado y se encuentra en el lado incorrecto del hiperplano.

Nos enfrentamos a un nuevo problema de optimización, donde seguimos queriendo maximizar el margen, pero ahora además queremos minimizar el error de clasificación:

$$\begin{aligned} \textbf{Función objetivo:} \quad & \min_{\mathbf{w},b,\xi_i} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \right\}, \\ \textbf{Restricciones lineales:} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \in \mathbf{D}, \quad \xi_i \geq 0. \end{aligned}$$

donde $C \geq 0$ es la *constante de regularización*, que nos permite controlar cómo influye la penalización de los errores a la función objetivo. Para valores de C grandes, se penalizarán muchos los errores, por lo que obtendremos márgenes más pequeños. Por el contrario, para valores de C pequeños, el hiperplano tendrá márgenes mayores a costa de ser más tolerante con las violaciones de margen.

De nuevo, nuestro **problema dual** asociado será el siguiente:

$$\begin{aligned} \textbf{Función objetivo:} \quad & \max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \right\}, \\ \textbf{Restricciones lineales:} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C, \quad \forall i = 1, ..., n. \end{aligned}$$

donde solo han cambiado las restricciones para α_i.

Una vez resuelto el problema dual, los parámetros $\mathbf{w}$ y b del problema primal vienen dados por las mismas expresiones que en la sección 3.2.4.1.

3.2.5. Problema no lineal

A pesar de que el clasificador SVM lineal es efectivo y suele ofrece resultados satisfactorios [25], se enfrenta a limitaciones en conjuntos de datos que distan mucho de ser linealmente separables.

Debemos tener claro que tanto el enfoque de margen duro como el de margen blando son problemas de optimización cuadrática convexa, con restricciones lineales. Esto implica que siempre existe una solución óptima global, la cual puede ser encontrada gracias a la naturaleza convexa de estos problemas. Sin embargo, la limitación de utilizar únicamente hiperplanos lineales puede ser demasiado restrictiva en algunos casos.

Un ejemplo ilustrativo es el de la figura 3.14, donde tenemos un conjunto de datos que es separable mediante una elipse, pero no a través una función lineal.

3.2.5.1. Características polinómicas

Una primera estrategia para trabajar con conjuntos de datos no lineales es añadir características adicionales, como términos polinómicos [26], similar a lo que se realiza en la regresión

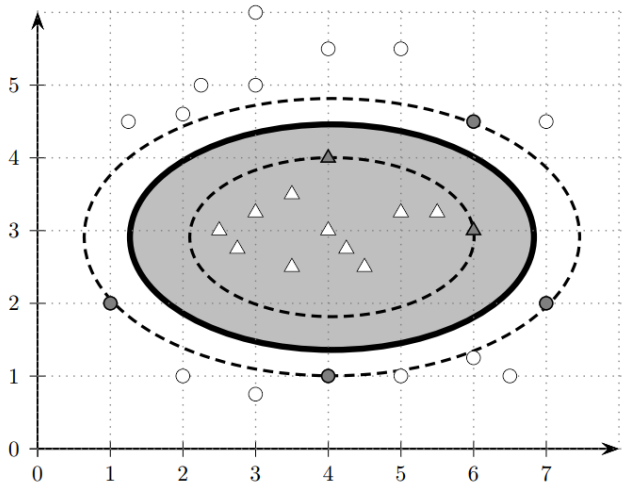


Figura 3.14.: Ejemplo de espacio no separable linealmente [3]

polinómica. En ciertas situaciones, este enfoque puede transformar los datos en un conjunto linealmente separable.

En la figura 3.15, observamos cómo el conjunto de datos de la izquierda, representado por una sola característica x_1, no linealmente separable, pues los datos de una clase están rodeados a izquierda y derecha por los datos de la otra clase. No obstante, si añadimos una segunda característica $x_2 = x_1^2$, el conjunto de datos se convierte en linealmente separable. Al trasladar los datos a un espacio de dimensión 2 mediante una transformación polinómica de grado 2, podemos encontrar fácilmente una separación lineal de estos.

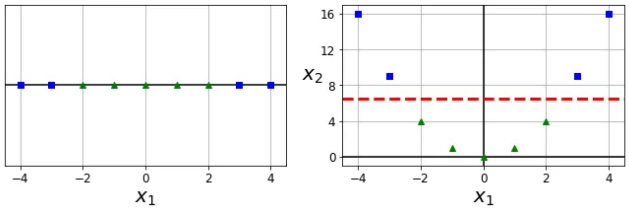


Figura 3.15.: Tranformación de los datos a un espacio de dimensión 2 [5]

Este planteamiento de añadir características polinómicas resulta sencillo de implementar, e incluso se puede extender a otros algoritmos de Aprendizaje Automático, pues estos términos polinómicos pueden ayudar al modelo a capturar relaciones complejas entre las características.

Sin embargo, este enfoque tiene sus limitaciones [26]. Por un lado, cuando se toma un grado polinómico pequeño, el método puede no ser suficiente para manejar conjuntos de datos con una estructura compleja. Por otro lado, para un grado polinómico alto, se genera una gran cantidad de nuevas características, lo cual puede aumentar la demanda computacional del modelo, ralentizando tanto las etapas de entrenamiento como de prueba. Además, esto puede conducir a problemas de sobreajuste, donde el modelo se ajusta demasiado a los datos de entrenamiento y pierde la capacidad de generalizar bien a nuevos datos.

3.2.5.2. SVM kernelizadas

Afortunadamente, una solución efectiva al problema anterior es la aplicación de una técnica matemática conocida como *truco del kernel* [26] [3] [5]. Este método nos aporta los beneficios de añadir características polinómicas, sin la necesidad de añadirlas explícitamente al modelo.

El truco del kernel consigue replicar el efecto de las características polinómicas, al operar sobre el espacio de características original mediante funciones kernel, que en un principio deberían transformar los datos a un espacio de mayor dimensión. Esto se realiza sin transformar realmente los datos a ese espacio, lo cual evita el alto coste computacional del que hablábamos en el punto anterior. Como resultado, el modelo puede manejar datos complejos y de alta dimensionalidad sin sufrir un aumento significativo en los requisitos computacionales o en el riesgo de sobreajuste.

Conceptualmente, la idea es transformar los puntos d-dimensionales x_i del espacio inicial, a puntos $\phi(x_i)$ en un espacio de características de alta dimensión, mediante alguna transformación no lineal ϕ. Es probable que los puntos $\phi(x_i)$ puedan ser separables linealmente en el nuevo espacio de características. El truco del kernel permite realizar todas las operaciones a través de la función de kernel calculada en el espacio de entrada, en lugar de tener que mapear los puntos al espacio de características.

Definición 3.29. Sea $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^n$ nuestro conjunto de datos d-dimensional. Sea $\mathcal{F}$ un espacio de dimensión superior, denominado espacio de características o *feature space*. Definimos la función $\phi : \mathbb{R}^d \to \mathcal{F}$ como **función de mapeo** o *feature map*, que hace corresponder cada punto x_i con un punto $\phi(x_i)$ de mayor dimensión, en $\mathcal{F}$.

Aplicando ϕ a cada punto, podemos obtener un nuevo conjunto de datos en el espacio de características $\mathcal{F}$, dado por $D_\phi = \{\phi(x_i), y_i\}_{i=1}^n$. Esta transformación permite que datos que no son linealmente separables en el espacio original puedan serlo en este nuevo espacio de características. Una vez se cumple esto, se pueden aplicar los métodos SVM lineales de margen duro y margen blando vistos anteriormente, para buscar un hiperplano de separación óptimo en el nuevo espacio.

De esta forma, el hiperplano de separación lineal sería:

$$h(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b = 0$$

y obtenemos el siguiente problema de optimización, para margen blando:

$$\begin{aligned} \textbf{Función objetivo:} \quad & \min_{\mathbf{w},b,\xi_i} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \right\}, \\ \textbf{Restricciones lineales:} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \in \mathbf{D}, \quad \xi_i \geq 0. \end{aligned}$$

Al igual que en el apartado anterior, a través de este problema primal podemos definir su problema dual asociado, dado por:

$$\textbf{Función objetivo:} \quad \max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \right\},$$

$$\textbf{Restricciones lineales:} \quad \sum_{i=1}^{n} \alpha_i y_i = 0,$$
$$0 \leq \alpha_i \leq C, \quad \forall i = 1, ..., n.$$

Entonces, la expresión del hiperplano óptimo quedaría:

$$h(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b = \sum_{i=1}^{n} \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle + b, \quad \mathbf{x} \in \mathbb{R}^d.$$

Observación 3.15. Observamos que la clase predicha para un nuevo punto $\mathbf{x} \in \mathbb{R}^d$ viene dada por

$$\hat{y} = sign(\sum_{i=1}^{n} \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle + b).$$

Podemos ver que para clasificar un punto mediante SVM, no es necesario calcular $\phi(\mathbf{x}_i)$ de manera aislada, sino que todas las operaciones se pueden realizar en términos de $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. Por ello, recurriremos al concepto de *kernel* en la siguiente sección.

3.2.5.3. Truco del kernel

Notemos que no es sencillo obtener una expresión de ϕ de forma que transformemos nuestro espacio de características inicial en un espacio linealmente separable. Por ello recurrimos al *truco del kernel*. El atractivo de las funciones kernel radica en su capacidad de realizar esta transformación de forma implícita, sin necesidad de definir ϕ, mediante el cálculo del producto interno dentro del espacio de características original.

Definición 3.30. Dado un par de puntos $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, definimos la **función kernel** como $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, dada por

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \phi(\mathbf{x})^T \phi(\mathbf{y})$$

El planteamiento del **problema de optimización** es igual que en la sección anterior, basta con sustituir el producto escalar por la función kernel:

$$\textbf{Función objetivo:} \quad \max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right\},$$

$$\textbf{Restricciones lineales:} \quad \sum_{i=1}^{n} \alpha_i y_i = 0,$$
$$0 \leq \alpha_i \leq C, \quad \forall i = 1, ..., n.$$

¿Cómo podemos caracterizar las funciones kernel? Nos encontramos ante el problema de verificar qué funciones cumplen la definición de función kernel. A veces, encontrar una ϕ que cumpla la definición no es nada trivial. El Teorema de Mercer nos aporta una **caracterización de las funciones kernel**. Para ello, antes necesitamos definir los siguientes conceptos [3]:

Definición 3.31. Una función $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ es **simétrica** si $k(x, y) = k(y, x), \forall x, y \in \mathbb{R}^d$.

Definición 3.32. Una función $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ es **semidefinida positiva** si verifica

$$\mathbf{c}^T \mathbf{K} \mathbf{c} \geq 0, \quad \forall \mathbf{c} \in \mathbb{R}^n, \quad \text{con } \mathbf{K} = (k(x_i, x_j))_{i,j=1}^n,$$

lo cual implica que

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0$$

para cualquier conjunto de puntos $x_1, x_2, ..., x_n \in \mathbb{R}^d$, y para cualquier conjunto de valores $c_1, c_2, ..., c_n \in \mathbb{R}$.

Proposición 3.1. *[3] Sea $k(x, z)$ una función simétrica definida en $\mathbb{R}^d$. Entonces $k(x, z)$ es una función kernel si y solo si la matriz*

$$\mathbf{K} = (k(x_i, x_j))_{i,j=1}^n,$$

es semidefinida positiva (es decir, tiene autovalores no negativos).

Teorema 3.5. ***(Teorema de Mercer)*** *[41] Sea X un subconjunto compacto[2] de $\mathbb{R}^n$. Supongamos que k es una función continua y simétrica tal que el operador $T_k : L^2(X) \to L^2(X)$[3], dado por*

$$(T_k f)(\cdot) = \int_X k(\cdot, \mathbf{x}) f(\mathbf{x})\, d\mathbf{x},$$

es positivo, es decir,

$$\int_{X \times X} k(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z})\, d\mathbf{x}\, d\mathbf{z} \geq 0,$$

para toda $f \in L^2(X)$. Entonces, podemos expandir $k(\mathbf{x}, \mathbf{z})$ en una serie uniformemente convergente[4], en términos de funciones $\phi_j \in L^2(X)$, tal que $\|\phi_j\|_{L^2} = 1$, de la siguiente forma:

$$k(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \phi_j(\mathbf{x}) \phi_j(\mathbf{z}).$$

La demostración puede encontrarse en [41].

Gracias al resultado anterior, podemos aportar una caracterización de las funciones kernel.

Teorema 3.6. ***(Caracterización de las funciones kernel)*** *[41] Sea $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ una función continua o con dominio finito. Sea ϕ una función de mapeo en un espacio de Hilbert F[5]. Entonces se puede descomponer en*

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle, \quad \mathbf{x}, \mathbf{z} \in \mathbb{R}^d$$

si y solo si k es finitamente semidefinida positiva (para cualquier subconjunto finito de $\mathbb{R}^d$).

La demostración puede encontrarse en [41].

[2] Ver definición B.2 en Glosario.
[3] Ver definición B.3 en Glosario.
[4] Ver definición B.4 en Glosario.
[5] Ver definición B.8 en Glosario.

3.2.5.4. Ejemplos de funciones kernel

Algunas de las funciones de kernel más utilizadas en la práctica incluyen ([26] [3]):

- **Kernel lineal**:
$$k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$$
Es la opción trivial, pues aplicar este kernel equivale a resolver el problema de optimización para espacios linealmente separables. Este kernel no implica un aumento de la dimensionalidad del espacio de características.

- **Kernel polinómico**:
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + c)^n$$
donde n es el grado del polinomio, y c una constante. Es una forma generalizada del kernel lineal ($n = 1$, $c = 0$). Este kernel, además de tener en cuenta las características de los puntos del espacio de entrada, también observa las combinaciones de estas. Para entenderlo con más claridad, veamos el siguiente ejemplo, extraído y simplificado de [3]:

  Ejemplo 3.4. Sean $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ dos puntos, y supongamos $c = 1$. Entonces nuestro kernel polinómico viene dado por la expresión
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2 = (x_1y_1 + x_2y_2)^2$$
  Como sabemos, la función kernel cumple $K(\mathbf{x}, \mathbf{y}) = \langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle$, por tanto buscamos ϕ que satisfaga
$$\langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle = (x_1y_1 + x_2y_2)^2,$$
  para $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$. Observamos que tomando
$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2),$$
  en efecto, se cumple
$$\begin{aligned}\langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle &= \langle(x_1^2, x_2^2, \sqrt{2}x_1x_2), (y_1^2, y_2^2, \sqrt{2}y_1y_2)\rangle \\ &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2 \\ &= (x_1y_1 + x_2y_2)^2 = k(\mathbf{x}, \mathbf{y})\end{aligned}$$

- **Kernel gaussiano**:
$$k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\|\mathbf{x} - \mathbf{y}\|^2)$$
donde γ determina la dispersión de la función, y por tanto el alcance de los vectores soporte sobre el modelo. Para valores de γ pequeños, los vectores soporte tendrán un gran alcance sobre el modelo y por tanto el modelo será no lineal. Por el contrario, para γ grandes, se producirán modelos más lineales. El kernel gaussiano es el más utilizado en la práctica [42], tanto en reconocimiento de patrones, como el de redes neuronales y otros ámbitos, consiguiendo resultados prometedores.

Observación 3.16. La función sigmoide $tanh(\mathbf{x}^T\mathbf{y} + b)$ a menudo se emplea en clasificación de SVM. Sin embargo, no se considera un kernel válido debido a que no siempre cumple las condiciones del Teorema de Mercer. En concreto, la matriz kernel puede no ser semidefinida positiva [43].

En la figura 3.16 podemos observar ejemplos de fronteras de decisión en SVM para los kernel más comunes.

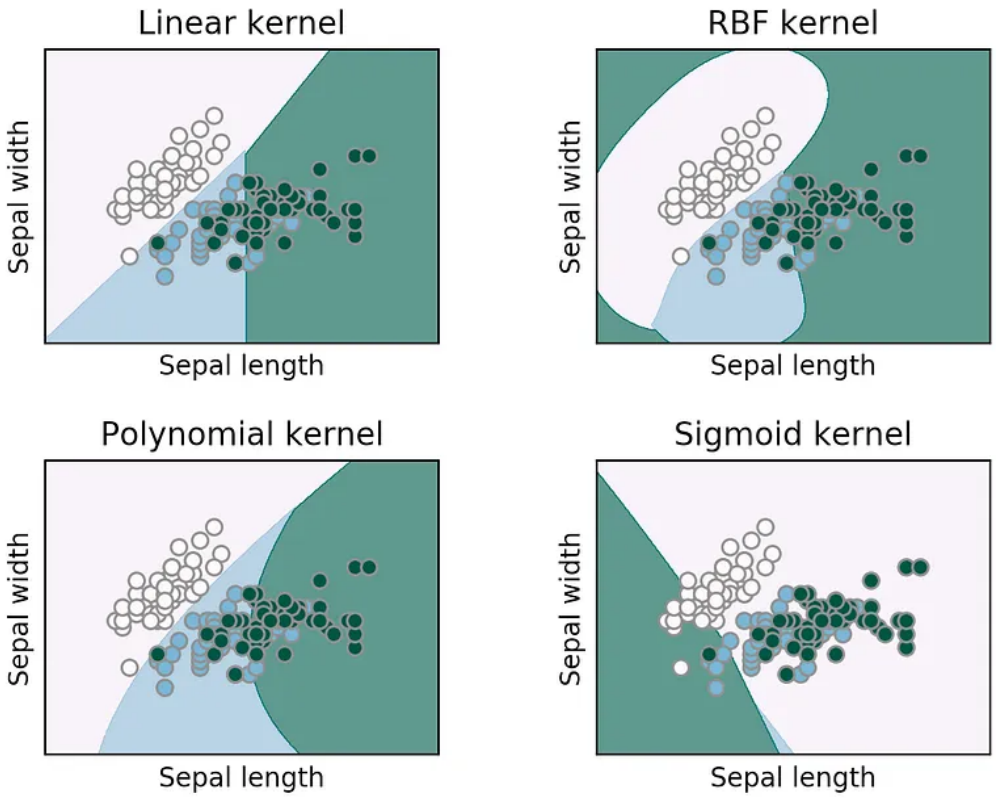


Figura 3.16.: Fronteras de decisión con distintos kernels para el dataset Iris [6]

Además de las anteriores, podemos crear nuevas funciones kernel mediante las siguientes operaciones:

Proposición 3.2. *[41] Sean k_1 y k_2 funciones kernels definidas sobre $\mathbb{R}^d \times \mathbb{R}^d$, sea $a \in \mathbb{R}_+$, $f : \mathbb{R}^d \to \mathbb{R}$, y B una matriz simétrica semidefinida positiva de $n \times n$. Entonces, dados los puntos $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$, las siguientes funciones son kernels:*

1. $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$,
2. $k(\mathbf{x}, \mathbf{z}) = ak_1(\mathbf{x}, \mathbf{z})$,
3. $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$,
4. $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,

5. $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T B \mathbf{z}$.

La demostración puede encontrarse en [41].

3.2.5.5. Implicaciones computacionales del uso de kernels

Hemos estudiado que en el ámbito de las Máquinas de Soporte Vectorial, el concepto de kernel desempeña un papel crucial, al permitir el manejo de problemas no lineales sin la necesidad de realizar transformaciones que pueden resultar muy costosas en términos computacionales.

Una de las principales ventajas de utilizar kernels es que la complejidad computacional depende del número de muestras y no del número de dimensiones del espacio de características, lo cual resulta muy interesante cuando ϕ mapea a un espacio de alta dimensión.

Además, mediante la selección de un kernel y unos parámetros adecuados, podemos conseguir capturar la estructura de los datos de manera más efectiva, mejorando así la capacidad de nuestro modelo para generalizar a nuevos datos.

4. Experimentación

Tras haber estudiado las bases matemáticas de dos de los algoritmos de aprendizaje supervisado más destacados, vamos a aplicar dichas técnicas a un caso real. El caso de estudio seleccionado para esta experimentación es el filtrado de correo electrónico *spam*, un problema que continua siendo pertinente en el ámbito de la seguridad informática y la gestión de datos.

Como se discutió en detalle en el capítulo 2, el filtrado de *spam* no solo facilita un manejo más sencillo del correo electrónico, sino que también protege a los usuarios de contenido potencialmente malicioso y fraudulento. La elección de este problema responde a la creciente necesidad de soluciones efectivas, ante la constante evolución de las técnicas empleadas por los *spammers*.

En este capítulo, aplicaremos los clasificadores seleccionados para abordar el filtrado de correos electrónicos, además de evaluar su efectividad en un conjunto de datos real. La experimentación incluirá una comparación detallada de los resultados obtenidos, analizando las métricas de rendimiento bajo diferentes configuraciones y parámetros. Además, aplicaremos técnicas más novedosas de preprocesado, como el uso de redes semánticas, para comprobar si se mejoran los resultados.

4.1. Conjunto de datos

Para la fase de experimentación, se ha elegido como corpus una base de datos textual extraída de Kaggle [44]. En esta se combinan mensajes de correo electrónico de tres fuentes distintas:

- **`LingSpam.csv`**: este conjunto de datos es uno de los más antiguos conocidos a nuestra disposición y está constituido por 2605 mensajes de *spam* y *ham* recuperados de Linguist List [45]. Estos mensajes se centran en ofertas de trabajo, oportunidades de investigación y discusiones sobre software.
- **`EnronSpam.csv`**: el conjunto de datos está formado por 10000 correos de varios trabajadores de la empresa Enron Corporation. El conjunto de datos original contiene aproximadamente 500000 emails y se puede encontrar en [46].
- **`SpamAssassin.csv`**: este conjunto de datos recoge 6046 mensajes donados por varios usuarios de un foro público. Al tratarse de usuarios variados, estos mensajes son de temas menos específicos que aquellos mensajes que podrían provenir de un mismo usuario. El conjunto de datos original se puede encontrar en [47].

Los conjuntos de datos que vamos a utilizar son simplificaciones de los originales, proporcionados por Kaggle. La decisión de coger mensajes de distintas fuentes se ha tomado para fomentar la diversidad de contenido y que el modelo esté entrenado sobre una mayor variedad de escenarios, con el fin de que pueda generalizar mejor ante nuevos datos. Por ejemplo, si sólo empleáramos la base de datos de EnronSpam, estaríamos limitando el espacio de detección del modelo al del ámbito empresarial.

Los tres archivos .csv constan de los siguientes atributos:

- `id:` identificador de cada instancia o mensaje.
- `Body:` cuerpo o contenido del mensaje.
- `Label:` clase a la que pertenece la instancia (toma el valor 1 si pertenece a Spam y 0 si pertenece a Ham).

4.2. Definición del problema

Nos encontramos pues ante un problema de **clasificación binaria** en un corpus textual, al cual vamos a aplicar varios algoritmos de aprendizaje supervisado para discernir qué técnica es capaz de identificar mejor las relaciones entre los contenidos de los mensajes.

Las posibles **etiquetas** en nuestro problema serán por tanto dos: **Spam** y **Ham**. Dividiremos el conjunto de datos en un **conjunto de entrenamiento o *train*** para entrenar y producir nuestro modelo, y un **conjunto de prueba o *test*** para constatar la eficacia del modelo y su capacidad para generalizar a datos nuevos con los que no ha sido entrenado. El objetivo de esta fase va a ser entrenar varios modelos que puedan distinguir con gran exactitud entre las clases Spam y Ham, tanto para el conjunto de entrenamiento como el de prueba.

4.3. Clasificación de textos

La tarea de clasificación de textos consiste en asignar una etiqueta a cada documento perteneciente a una colección, la cual indica a qué clase pertenece dicho documento. La decisión sobre qué etiqueta debe asignarse se toma a partir de un modelo construido mediante un conjunto de entrenamiento, que no es más que una parte de la colección donde los documentos tienen ya asignados una etiqueta. El objetivo es la construcción de un modelo que prediga correctamente la clase de nuevos documentos, llamado conjunto de prueba, los cuales no deben intervenir en la construcción del modelo para evitar cometer un sesgo. Cuando deseamos mejorar la calidad o eficacia de la clasificación para un dominio concreto, podemos aplicar dos enfoques [27]:

- **Modificar el modelo predictivo:** ya sea cambiando el clasificador elegido por otro, o alterando los hiperparámetros, estudiando qué combinación provee de mejores resultados.
- **Modificar la representación de los datos:** la eficacia de los clasificadores puede variar significativamente dependiendo de cómo se representen los datos. Elegir una representación adecuada para los datos es un desafío en sí mismo. Las formas más comunes de representar un documento incluyen el **modelo vectorial**, donde cada documento se representa como un vector cuyas dimensiones corresponden al tamaño del vocabulario y cuyos atributos reflejan la frecuencia de cada término en el documento. También se utilizan representaciones binarias, donde se indica simplemente la presencia o ausencia de términos, y otros esquemas de ponderación como **TF-IDF** (Frecuencia de Término - Frecuencia Inversa de Documento), que ajusta la frecuencia de los términos por su

rareza en la colección de documentos, proporcionando más peso a los términos más distintivos.

Además de los enfoques que se comentan en [27], no debemos olvidar que el **preprocesamiento de los datos** juega un papel crucial en la clasificación de textos. Este incluye técnicas de transformación de los datos y de Procesamiento del Lenguaje Natural [11], como la normalización de texto (por ejemplo, convertir el texto a minúsculas), la eliminación de palabras vacías, la tokenización, la lematización o el stemming. Cada uno de estos pasos puede ayudar a reducir el ruido y la dimensionalidad de los datos, lo que puede mejorar significativamente la precisión del modelo clasificador.

4.4. Visualización y análisis exploratorio

El primer paso para implemetar un clasificador de *spam* con éxito es conocer los datos con los que estamos trabajando. Para ello nos vamos a apoyar en diversas técnicas de análisis de datos, con el fin de identificar patrones, anomalías y características clave que puedan influir en el rendimiento del modelo de clasificación. Este análisis preliminar es crucial para diseñar estrategias de preprocesamiento de datos efectivas y posteriormente seleccionar las técnicas de modelado más adecuadas. Esta implementación se encuentra en el cuaderno `Visualización.ipynb`.

4.4.1. Limpieza y representación de los datos

Dentro de este preprocesamiento inicial, previo a la visualización de los datos, vamos a aplicar las siguientes técnicas para limpiar y normalizar los datos textuales en nuestro conjunto de datos:

1. **Eliminar valores duplicados**: Para asegurar que nuestro análisis sea sobre datos únicos, utilizamos la función `drop_duplicates()` (Apéndice B.2.1). Esta función detecta y elimina cualquier instancia repetida en el dataset, necesario para mantener la calidad de los datos y evitar sesgos debidos a repetición de información.

2. **Eliminar valores nulos**: Empleamos una expresión regular que reemplaza cadenas vacías que consisten únicamente en espacios en blanco por `NaN`. Esta operación es seguida por la eliminación de todas las filas que contengan valores `NaN` en la columna `Body` mediante `dropna()` (Apéndice B.2.1). Esta limpieza es fundamental para no distorsionar el análisis con textos que no contienen información útil.

3. **Eliminar mayúsculas**: Convertimos todo el texto a minúscula para uniformizar el formato y facilitar el procesamiento de texto. Esto es importante porque en el análisis textual, las palabras en mayúsculas y minúsculas se consideran distintas por defecto. Esto reduce la dispersión de variantes que pueda tener una misma palabra, simplificando el análisis. Lo conseguimos mediante `str.lower()` (Apéndice B.2.1).

4. **Identificación de tokens**: Identificamos enlaces o URLs mediante una expresión regular y los sustituimos por el token 'URL'. Este enfoque ayuda a estandarizar las referencias a recursos externos en el texto y simplifica la representación del texto, al reducir la variabilidad que las URLs podrían introducir.

5. **Eliminar contracciones propias del idioma**: como estamos trabajando con un conjunto de datos en inglés, es pertinente identificar las contracciones para que al eliminar los signos de puntuación no queden palabras sueltas. Para ello, hemos definido un diccionario con las contracciones más comunes y las reemplazamos por su expresión equivalente.

6. **Eliminar signos de puntuación**: Eliminamos todos los signos de puntuación y los reemplazamos por un espacio vacío. Esto ayuda a reducir el ruido en el texto y permite que los modelos se enfoquen en las palabras y su significado sin distracciones causadas por caracteres especiales. Hacemos uso del conjunto de signos de puntuación `string.punctuation` (Apéndice B.2.3).

7. **Eliminar caracteres no alfabéticos**: Utilizamos un patrón de expresiones regulares para eliminar cualquier caracter que no sea alfabético, incluyendo números y símbolos, utilizando `re.sub` (Apéndice B.2.4). Esto mejora la uniformidad del texto y facilita la identificación de términos clave.

8. **Eliminar palabras vacías**: Las palabras vacías o *stop words* (pronombres, preposiciones, conjunciones, etc.) se deben eliminar del texto antes de la tokenización, para concentrar el análisis en palabras que aportan más significado al contenido. Para ello, scikit-learn proporciona una lista predeterminada de palabras vacías en inglés, denominada `ENGLISH_STOP_WORDS` (Apéndice B.2.2). Esta lista se puede actualizar con palabras adicionales según sea necesario. Eliminar estas palabras vacías también reduce la dimensionalidad del dataset.

9. **Eliminar términos sueltos**: los términos de longitud menor que dos, como números o palabras sueltas, no ayudan a la detección de *spam* pero sí pueden dificultar la detección de patrones o de términos importantes en la etapa de Visualización.

10. **Tokenización**: utilizamos la función `word_tokenize()` de la librería de PLN (Apéndice B.2.5), para convertir nuestro conjunto de datos en una lista de listas de términos, donde cada instancia es representada por una lista de términos. Así conseguimos una estructura de datos en forma de términos separados, gracias al cual podremos hacer un análisis sobre la distribución del conjunto de datos, como estudiar las palabras más frecuentes.

Estas técnicas transforman el conjunto de datos, preparando su estructura y su contenido para un análisis más robusto. A partir de estas técnicas obtendremos dos DataFrames:

- `data_clean`: le aplicamos a nuestro conjunto de datos las primeras nueve técnicas anteriores. De esta forma podemos pasar de una estructura repleta de símbolos, caracteres especiales, enlaces, etc., a otra más limpia después de transformar los datos, como mostramos en la tabla 4.1 para tres ejemplos extraídos de nuestro conjunto de datos.

- `data_tokenized`: además de las técnicas anteriores, le aplicamos una tokenización a nuestro conjunto de datos. En la tabla 4.2 observamos cómo quedan las instancias tras ser tokenizadas. El fin detrás de esto es poder aplicar algunas técnicas de visualizado de forma más práctica.

De esta forma obtenemos dos DataFrames para realizar nuestro análisis de datos: `data_clean` que contiene los datos limpios pero en forma de string, y `data_tokenized` que contiene los datos tokenizados en forma de lista de términos.

Label	Texto original	Texto limpio
Spam	1) Fight The Risk of Cancer! http://www.adclick.ws/p.cfm?o=315&s=pk007 2) Slim Down - Guaranteed to lose 10-12 lbs in 30 days http://www.adclick.ws/p.cfm?o=249&s=pk007	fight risk cancer URL slim guaranteed lose lbs days URL
Spam	Subject: make $3500 per week using your home computer! put my free software in your computer... start making huge amounts of cash... without working!!!	make week using home computer free software computer start making huge amounts cash working
Ham	Subject: intern compensation sally : summer analysts will benefit from our recent compensation changes. their monthly salary this summer will be $3333.00 please let me know if you have any questions	intern compensation sally summer analysts benefit recent compensation changes monthly salary summer let know questions

Tabla 4.1.: Comparación de correos antes y después de la limpieza

Label	Texto limpio	Texto tokenizado
Spam	fight risk cancer URL slim guaranteed lose lbs days URL	['fight', 'risk', 'cancer', 'URL', 'slim', 'guaranteed', 'lose', 'lbs', 'days', 'URL']
Spam	make week using home computer free software computer start making huge amounts cash working	['make', 'week', 'using', 'home', 'computer', 'put', 'free', 'software', 'computer', 'start', 'making', 'huge', 'amounts', 'cash', 'working']
Ham	intern compensation sally summer analysts benefit recent compensation changes monthly salary summer let know questions	['intern', 'compensation', 'sally', 'summer', 'analysts', 'benefit', 'recent', 'compensation', 'changes', 'monthly', 'salary', 'summer', 'let', 'know', 'questions']

Tabla 4.2.: Comparación de textos limpios y tokenizados

4.4.2. Análisis estadístico descriptivo

En primer lugar realizamos un **análisis cuantitativo** de los correos en nuestro conjunto de datos. Para ello calculamos la longitud de cada instancia, en términos de número de caracteres, en número de palabras y número de oraciones.

- **Número de caracteres**: Se calcula la longitud de cada texto, es decir, el número total de caracteres que contiene. Para ello se usa la función `len`, que devuelve la longitud de una cadena en Python.
- **Número de palabras**: Para calcular el número de palabras en cada texto, se utiliza la función `word_tokenize()` de NLTK. Esto divide el texto en palabras, y luego contamos el número de elementos resultantes.
- **Número de frases**: La función `sent_tokenize()` de NLTK se emplea para dividir el texto en oraciones.

A continuación se llama a la función `describe()` de Pandas (Apéndice B.2.1), que proporciona un resumen estadístico que incluye datos como la media aritmética de las longitudes, la desviación estándar, valores mínimo y máximo y los percentiles.

Esta información nos puede ayudar a entender la distribución y la variabilidad de los datos, para identificar patrones y anomalías. En la figura 4.1 podemos ver el resumen estadístico para cada clase.

	num_characters	num_words	num_sentence
count	11010.00	11010.00	11010.00
mean	2137.50	283.57	1.00
std	98451.82	12666.99	0.01
min	0.00	0.00	0.00
25%	255.00	37.00	1.00
50%	557.00	80.00	1.00
75%	1213.00	166.00	1.00
max	10327244.00	1328682.00	1.00

(a) Resumen estadístico de la clase Ham

	num_characters	num_words	num_sentence
count	6560.00	6560.00	6560.00
mean	1020.17	138.98	1.00
std	2314.87	310.56	0.02
min	0.00	0.00	0.00
25%	234.00	34.00	1.00
50%	440.00	62.00	1.00
75%	979.25	136.00	1.00
max	111115.00	15132.00	1.00

(b) Resumen estadístico de la clase Spam

Figura 4.1.: Comparación de resúmenes estadísticos

En general, hemos obtenido unos resultados bastante contrastantes. Por un lado, el número de palabras medio en instancias Ham es de 284, mientras que en instancias Spam es de 138 (la mitad). Además, la desviación de estos datos es mucho mayor en Ham que en Spam. Por tanto, podemos concluir que en general los correos Ham son más largos y menos homogéneos. Por esta razón, la cantidad de palabras podría ser un indicativo de qué tipo de correo estamos tratando.

Por otro lado, observamos que en ambos casos hay correos con unas longitudes desorbitadas, los cuales podríamos considerar anomalías.

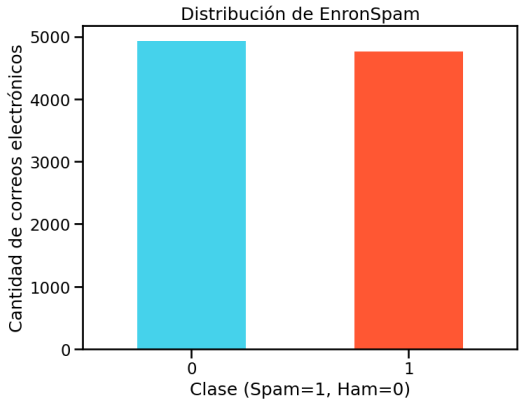
4.4.3. Distribución de las clases

En esta sección exploraremos cómo están distribuidas las dos clases dentro del conjunto de datos, tarea esencial para entender la naturaleza del problema. Analizar la distribución de las clases nos puede ayudar a identificar si existe un desbalanceo significativo con respecto a alguna de las características del conjunto de datos, que podría influir en el rendimiento del modelo de clasificación. Una distribución equitativa de las clases permite un entrenamiento más efectivo y menos sesgado de los algoritmos de aprendizaje automático.

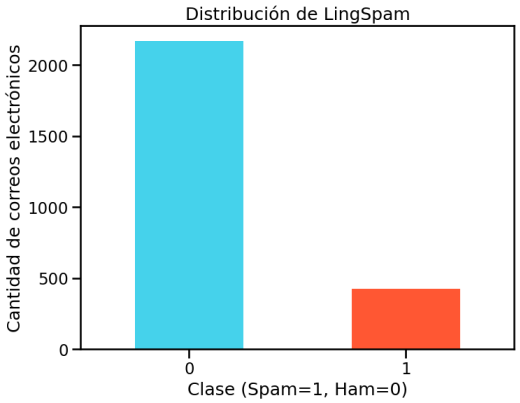
El desbalanceo de clases ocurre cuando las proporciones de las clases objetivo en un conjunto de datos no son iguales o están significativamente inclinadas hacia una clase. Este desbalanceo puede conducir a modelos con un rendimiento pobre en la clase minoritaria, lo cual es un aspecto crítico en la detección de *spam*, donde la clase Spam suele ser precisamente la minoritaria.

En la figura 4.5, podemos visualizar la distribución de las clases mediante gráficos de barra,

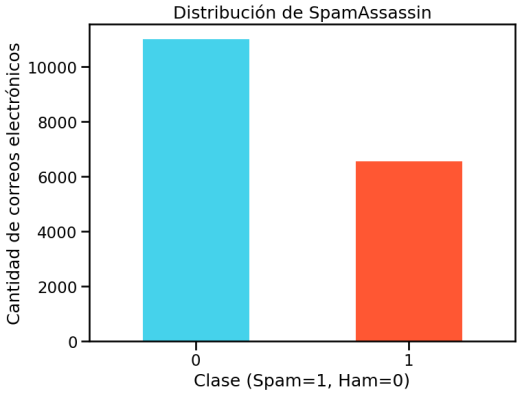
para los tres conjuntos de datos. Observamos que la primera distribución tiene un número de instancias bastante parejo, al contrario de lo que podemos observar en las dos siguientes.



(a) Distribución de EnronSpam



(b) Distribución de LingSpam



(c) Distribución de SpamAssassin

Figura 4.2.: Gráficos de barras con la distribución de clases

En la figura 4.3, podemos observar un gráfico circular o *pie chart*, con la proporción del conjunto de datos general. En efecto, el número de instancias Ham casi triplica el de Spam.

Nos encontramos ante un problema de desbalanceo de clases, que puede mermar la eficiencia de los clasificadores. Los modelos entrenados con datos desbalanceados pueden desarrollar un sesgo hacia la clase más frecuente, lo que resulta en una precisión pobre al predecir la clase minoritaria, en este caso la clase Spam. Por tanto, concluimos que necesitaremos aplicar una técnica de balanceo en la fase de Preprocesado.

4.4.4. Longitud de los correos

Gracias a los datos recopilados en el análisis estadístico (el número de palabras por instancia), estudiamos ahora la relación entre la clase y la cantidad de palabras de cada instancia. Para

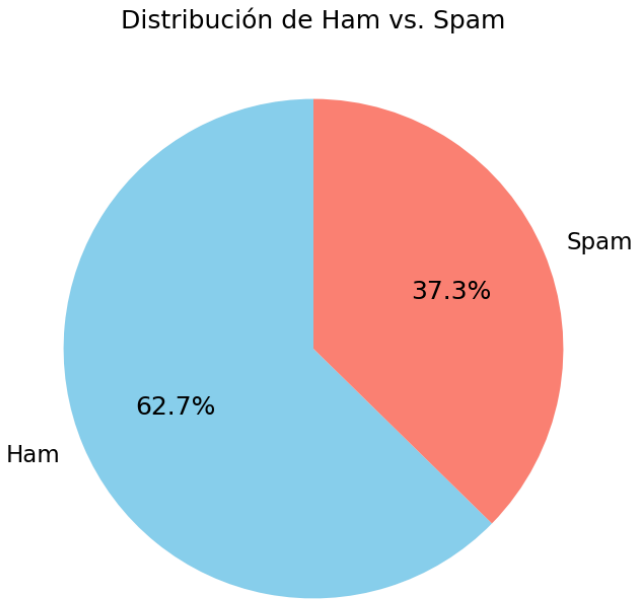


Figura 4.3.: Proporción total de las instancias

ello, visualizamos en primer lugar un histograma para el conjunto de datos general. Dejamos fuera el 2 % de instancias a la derecha, para evitar los *outliers*.

Observamos en la figura 4.4 la distribución de la cantidad de palabras es similar para ambas clases. Hay que tener en cuenta que el conjunto de Spam es menor que el de Ham en dos de los datasets, pero su distribución es prácticamente igual. Los correos Spam suelen tener ligeramente mayor numero de palabras, pero no encontramos nada significativo.

Vamos a representar el mismo histograma pero para cada conjunto de datos por separado. Ignorando el desbalanceo del que ya hemos hablado en la sección 4.4.3, el número de palabras sigue distribuciones similares.

4.4.5. Distribución de términos

Una vez hecho el análisis de la estructura del conjunto de datos, procedemos a analizar el contenido de este. Para ello, visualizaremos, en primer lugar, un *word cloud* o nube de palabras. Posteriormente representaremos los unigramas, bigramas y trigramas de términos.

Antes de realizar esta visualización hacemos el siguiente inciso: Al visualizar por primera vez las nubes de palabras y los n-gramas, observamos que el texto todavía no estaba bien limpio. Por ello, realizamos las siguientes correcciones en la limpieza del texto, que ya se encuentran incluidas en la sección 4.4.1, en los puntos 5, 7 y 9:

- **Eliminación de contracciones:** al representar los bigramas, vimos que había palabras como “don’t” que al aplicar la función de limpieza, resultaba como “don” “t”. Esto lo

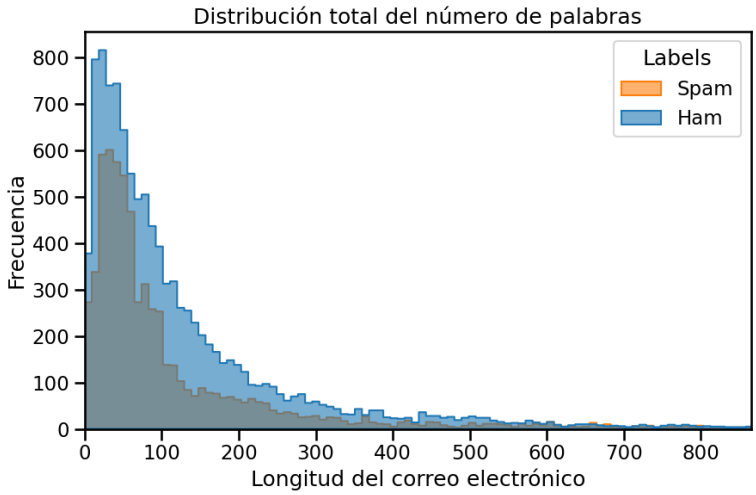
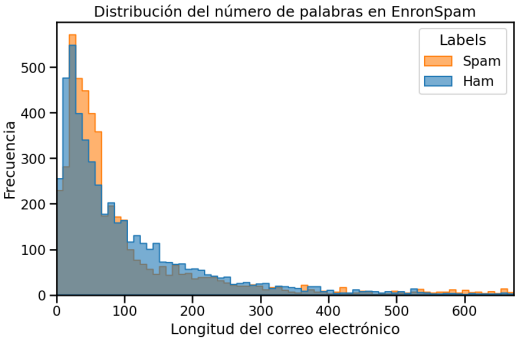
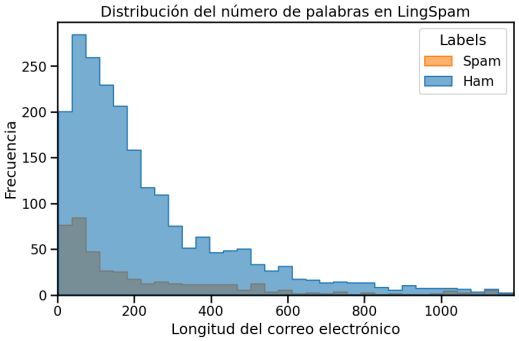


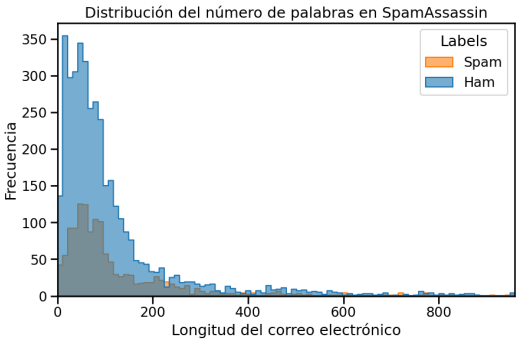
Figura 4.4.: Distribución total de la longitud de las instancias



(a) Distribución de la longitud en EnronSpam



(b) Distribución de la longitud en LingSpam



(c) Distribución de la longitud en SpamAssassin

Figura 4.5.: Distribución de la longitud en palabras para cada clase

hemos arreglado sustituyendo las contracciones del inglés por su expresión equivalente, por lo que en el ejemplo anterior quedaría “do” “not”.

- **Eliminación de caracteres no alfabéticos:** en un principio nos quedábamos con todos los caracteres alfanuméricos. Sin embargo, vimos que quedaban mucho números y cantidades sueltos, que no aportaban gran cosa sin un contexto, por lo que hemos decidido quedarnos únicamente con los términos formados por caracteres alfabéticos.
- **Eliminar términos con longitud baja:** eliminamos los términos con longitud menor que 2 pues eran muy comunes y ensuciaban los n-gramas, y por tanto podrían afectar al rendimiento del modelo.

Las siguientes visualizaciones sobre el contenido del conjunto de datos, se realizaron después de haberse aplicado dichas correcciones.

4.4.5.1. Word Clouds

Un *word cloud* o nube de palabras (Ver Apéndice B.2.6) consiste en una representación gráfica de la frecuencia de las palabras en un texto. Las palabras más frecuentes aparecen más destacadas, con un tamaño mayor, mientras que las menos frecuentes son más pequeñas.

Esta herramienta ayuda a percibir rápidamente cuáles son los términos más predominantes en un texto. Gracias a esto, decidiremos qué términos pueden ser determinantes a la hora de clasificar un mensaje como Spam o Ham. Las palabras que destacan en una nube de palabras de correos Spam pueden indicar patrones de lenguaje típicos de estos mensajes, como por ejemplo términos relacionados con ofertas, urgencia, o peligro. De manera similar, los términos más comunes en mensajes Ham pueden reflejar temas cotidianos, conversacionales o empresariales.

Además de lo anterior, una nube de palabras puede servir como una herramienta de diagnóstico preliminar para ajustar la clasificación de texto. Por ejemplo, si algunas palabras irrelevantes aparecen frecuentemente en la imagen, sería indiciativo de que necesitamos ajustar los filtros de stop words o de refinar las técnicas de preprocesamiento.

En la figura 4.6 podemos visualizar las nubes de palabras para Ham y Spam, en el conjunto de datos general. Estas muestran claramente algunas diferencias en los términos más frecuentes.

En correos Ham (legítimos), algunas de las palabras más visibles son “language”, “university”, “enron”, “linguistic”, etc. Esto sugiere que el contenido de los correos está relacionado con el trabajo y el mundo académico. Otras palabras un poco menos comunes pero también destacables son “URL”, “mail”, “program”, “information”, “new”, lo que indica que estos correos también pueden estar relacionados con el mundo de la tecnología.

En correos Spam (no deseados), destacan términos como “free”, “email”, “money”, “report”, “business”, “order”, “click”, etc., lo cual indica relación con el marketing y la publicidad. La palabra “free” es particularmente destacada, pues es una táctica común en correos Spam para atraer la atención. Por otro lado, la palabra “email” es común en este contexto por lo que es normal que aparezca tanto, para ambas clases.

A pesar de lo que podríamos haber supuesto, el término “URL” no destaca demasiado en los correos Spam. Esto puede ser porque, como es bien sabido, los *spammers* a menudo

Figura 4.6.: Word clouds de las dos clases

utilizan técnicas para ocultar o disfrazar las URLs para evitar filtros de spam, y que por tanto han podido pasar desapercibidos en nuestra limpieza de URLs. Esto puede incluir:

- El uso de espacios o caracteres especiales que no están cubiertos por nuestro patrón.
- Encurtamiento de URLs que no comienzan con `http://` o `https://`.
- Incorporación de texto o caracteres dentro de la URL para hacerla menos detectable.

Hemos pensado alguna solución como diferenciar entre URLs que utilizan el protocolo HTTPs y otras que no. Sin embargo, no hemos conseguido resultados destacables.

Por tanto, lo que vamos a hacer únicamente es importar una biblioteca especializada en detección de URLs en texto, como `URLExtract` (Apéndice B.2.8). Estas bibliotecas suelen tener patrones más completos y actualizados para la detección de URLs.

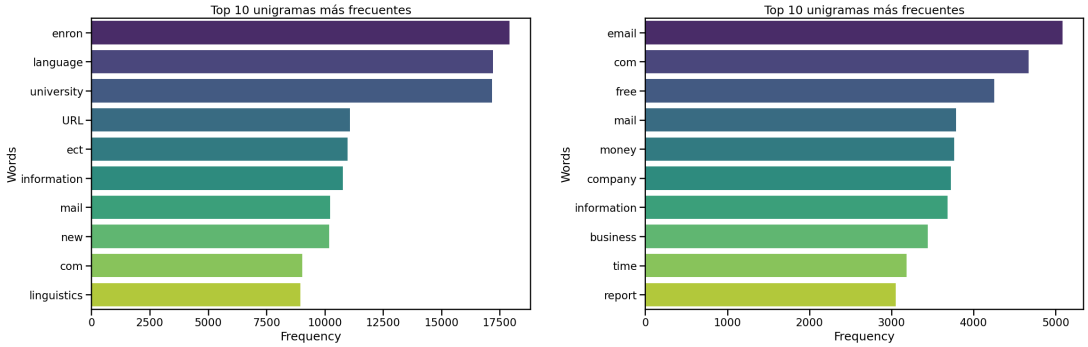
En general, hemos visto que los correos Spam utilizan un lenguaje que incluye términos de marketing y transacciones, mientras que los Ham tienden a incluir un vocabulario sobre la comunicación académica y profesional. Esto puede ser utilizado para mejorar los filtros de spam y los algoritmos de clasificación.

Una idea clave es que podríamos utilizar estas diferencias en la frecuencia de palabras a la hora de entrenar modelos de clasificación. Algunas técnicas como TF-IDF pueden ser útiles aquí para ponderar las palabras en función de su importancia relativa en cada clase.

4.4.5.2. n-gramas

Además de las nubes de palabras, hemos considerado interesante visualizar algunos **n-gramas**. Estos son básicamente diagramas de conteo de frecuencias de palabras, para ver por ejemplo qué 10 términos son más comunes. Hemos visualizado tres tipos de n-gramas:

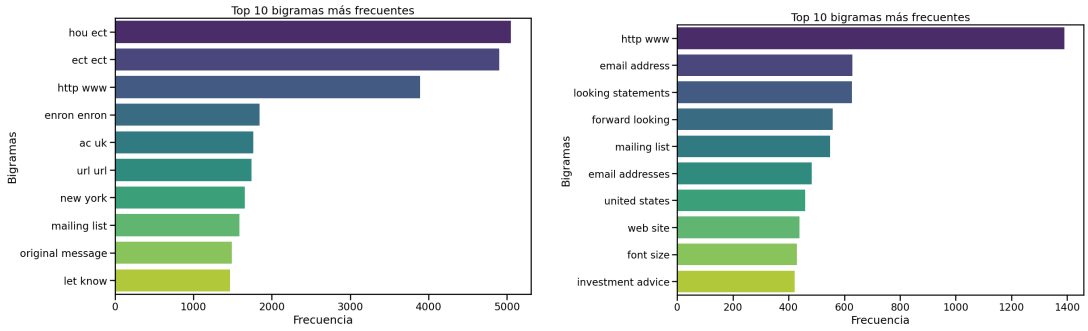
- **Unigramas:** utilizamos la función `Counter` (Apéndice B.2.7) para contar la frecuencia de términos. En la figura 4.7 podemos observar que, como ya vimos con los *word clouds*, en los correos Ham, palabras como "enron", "language", y "university" son frecuentes, lo cual sugiere un contexto más profesional y académico. En los correos Spam, predominan palabras como "email", "free", y "money", lo que indica un enfoque en ofertas y ganancias rápidas.
- **Bigramas:** usamos la función `bigrams` (Apéndice B.2.5) para agrupar pares términos, y la función `Counter` para contarlos. En la figura 4.8, podemos ver que en los correos Ham predominan secuencias como "hou ect" y "ect ect", que no nos aportan nada, y otras relacionadas con empresas. Por otro lado, en los Spam, frases como "http www" y "email addresses" aparecen frecuentemente, las cuales sugieren intentos de direccionar a los usuarios a sitios web o recolectar información personal.
- **Trigramas:** usamos la función `trigrams` (Apéndice B.2.5) para agrupar grupos de tres términos, y la función `Counter` para contarlos. En la figura 4.9 observamos que en los correos Ham, trigramas como "corp enron enron" y "linux users group" sugieren discusiones sobre negocios y tecnología. En los correos Spam, combinaciones como "forward looking statements" y "pills pills pills" son indicativas de contenido promocional y a menudo engañoso.



(a) Unigramas para la clase Ham

(b) Unigramas para la clase Spam

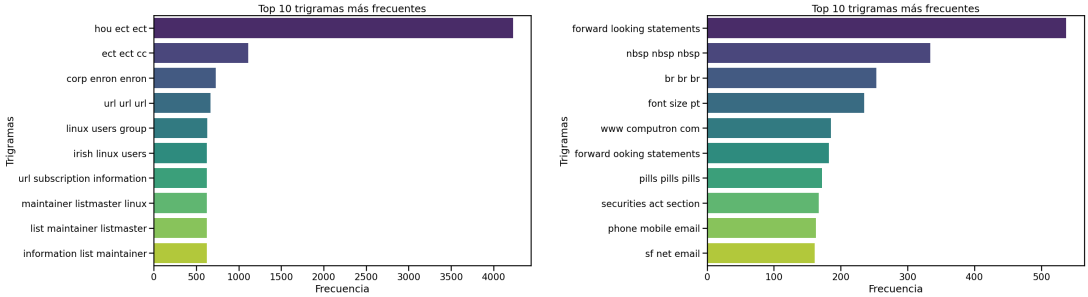
Figura 4.7.: Comparación de unigramas



(a) Bigramas para la clase Ham

(b) Bigramas para la clase Spam

Figura 4.8.: Comparación de bigramas



(a) Trigramas para la clase Ham

(b) Trigramas para la clase Spam

Figura 4.9.: Comparación de trigramas

4.5. Preprocesado

Antes de entrenar el clasificador, es crucial realizar el preprocesado de los datos para garantizar la máxima eficacia del algoritmo de aprendizaje automático seleccionado. Hemos dividido el preprocesado en dos etapas principales: *data cleaning* y *data preprocessing*.

Primero, el preprocesado inicial (*data cleaning*), realizado en 4.4.1, ha sido necesario para

poder realizar una visualización en condiciones de los datos. Este paso se centró en limpiar los datos de cualquier inconsistencia o imperfección que pudiera distorsionar el análisis visual y estadístico.

Tras aplicar diversas técnicas de visualización a nuestros datos, nos hemos percatado de varias tendencias, patrones y características presentes en nuestro dataset. Esta comprensión nos permite realizar un segundo nivel de preprocesado (*data preprocessing*), donde vamos a sacarle el máximo provecho al conocimiento obtenido. En esta sección, ajustamos y transformamos los datos de manera que se alineen mejor con los requisitos y la naturaleza del modelo de aprendizaje automático que vamos a emplear.

4.5.1. Selección de características. Filtrado de palabras poco frecuentes

Hemos optado por una estrategia de filtrado manual, para controlar el vocabulario utilizado en nuestros modelos. Esta estrategia consiste en eliminar palabras que aparecen menos de cinco veces en todo el corpus. Realizamos un recuento de la frecuencia de cada palabra en el conjunto de datos completo, con `Counter()` (Apéndice B.2.7), y posteriormente filtramos aquellas palabras que no alcanzan el umbral mínimo establecido. De esta manera conseguimos ajustar aquellas características que consideramos relevantes para nuestro modelo, reduciendo la dispersión de la matriz de características y eliminando términos poco informativos que podrían afectar el rendimiento del modelo.

4.5.2. Balanceo de clases

En la fase de Visualización vimos que es preciso realizar un balanceo de las clases para evitar sesgos, pues el modelo se adaptará a los ejemplos de la clase mayoritaria sin detectar correctamente los patrones de la clase minoritaria. Las distintas técnicas que hemos considerado son:

- **Sobremuestreo de la clase minoritaria:** Consiste en aumentar el número de instancias en la clase minoritaria replicándolas hasta alcanzar un balance más equitativo.
- **Submuestreo de la clase mayoritaria:** Esta técnica reduce el número de instancias en la clase mayoritaria para igualar la cantidad de la clase minoritaria.
- **Generación sintética de muestras (SMOTE):** SMOTE crea instancias sintéticas de la clase minoritaria en lugar de crear copias exactas, ayudando a proporcionar más variabilidad. Sin embargo, también introduce el riesgo de generar datos que no representan adecuadamente la realidad.

Finalmente hemos optado por aplicar sobremuestreo, pues nos permite mantener toda la información de la clase mayoritaria mientras aumentamos la representación de la clase minoritaria, evitando la pérdida de datos potencialmente valiosos que ocurre en el submuestreo.

Para ello hacemos uso de `RandomOverSampler()` de la biblioteca `imbalanced-learn` B.2.9. `RandomOverSampler()` va replicando aleatoriamente las instancias de la clase minoritaria hasta alcanzar un equilibrio deseado.

4.5.3. Vectorización

El objetivo de la vectorización es convertir el cuerpo de cada correo electrónico en una representación numérica, donde cada palabra se convierte en una característica y se cuenta su

frecuencia en cada correo electrónico.

Para ello utilizaremos dos vectorizadores: la función `CountVectorizer()` y la función `TfidfVectorizer()`, ambas de `sklearn` (Apéndice B.2.2).

Realizamos este proceso porque el clasificador de Naive Bayes Multinomial está pensado para aplicarlo en clasificación con características discretas, como recuentos de palabras [37]. Los datos de entrenamiento y de prueba se deben pasar al clasificador bien como una matriz densa (array-like) o una matriz dispersa (sparse matrix).

Para el clasificador SVM, podemos comprobar en la documentación que también necesitamos pasarle una matriz densa o dispersa [48].

Una matriz densa es aquella en la que la mayoría de sus elementos son distintos de cero. En cambio, una matriz dispersa se caracteriza por tener una gran cantidad de elementos que son cero. Mientras que las matrices densas almacenan cada uno de los elementos en la memoria independientemente de su valor, las matrices dispersas almacenan solo los elementos distintos de cero y sus índices de filas [49] [50]. Por este motivo, el uso de matrices dispersas permite reducir de forma significativa la cantidad de memoria que se necesita para almacenar datos.

Las funciones `CountVectorizer()` y `TfidfVectorizer()` precisamente devuelven una matriz dispersa, que ocupa menos espacio en memoria que una matriz densa. Hemos elegido esto porque el conjunto de datos, al representar apariciones de palabras, es muy disperso, lo que significa que la mayoría de entradas en la matriz serán ceros.

- `CountVectorizer()` convierte una colección de documentos de texto en una matriz de recuentos de tokens. Simplemente cuenta cuántas veces aparece cada palabra en cada documento. Esto resulta en una matriz donde cada entrada i,j es el número de veces que la palabra j aparece en el documento i.

  Este método tiene como ventaja que es simple y directo, y la frecuencia absoluta de las palabras puede ser un buen indicador de su relevancia en el documento. Su principal inconveniente es que no tiene en cuenta la importancia relativa de las palabras a través de los documentos. Palabras comunes como preposiciones o palabras de parada que aparecen en muchos documentos pueden distorsionar la representación sin aportar mucha información sobre el contenido real del texto.

- `TfidfVectorizer()` convierte textos a una matriz de características TF-IDF (*Term frequency – Inverse document frequency*, Frecuencia de término - Frecuencia inversa de documento). Este método no solo cuenta la frecuencia de las palabras, sino que también pondera estos recuentos según la importancia de la palabra en el conjunto de documentos. La importancia aumenta proporcionalmente al número de veces que aparece una palabra en un documento pero se compensa por la frecuencia de la palabra en el corpus.

  Una ventaja con respecto al vectorizador anterior es que al ponderar las palabras, TF-IDF puede ofrecer una mejor representación de la importancia de cada palabra para cada documento. Esto puede ser especialmente útil para distinguir documentos relevantes o temas dentro de grandes colecciones de texto. A pesar de su utilidad, TF-IDF puede ser menos efectivo en casos donde las palabras que aparecen con mucha

frecuencia en una categoría pero también en todo el corpus pueden ser penalizadas excesivamente.

¿Qué método usar para cada clasificador?

- **Naive Bayes Multinomial:** este clasificador funciona bien con CountVectorizer porque se basa en las probabilidades de las frecuencias de las palabras [51]. Los recuentos de palabras proporcionan una buena base para estimar las probabilidades de que un documento pertenezca a una clase o a otra, lo que hace que el CountVectorizer sea muy compatible con Naive Bayes, especialmente cuando el tamaño del vocabulario y la cantidad de datos son adecuados.
- **SVM:** este clasificador puede beneficiarse más de TfidfVectorizer [52] ya que este vectorizador reduce la influencia de palabras comunes que no son muy útiles para la clasificación. Esto puede ayudar a SVM a establecer un hiperplano de decisión más efectivo y preciso en un espacio de características donde las dimensiones (palabras) más informativas reciben mayor peso.

A pesar de que en un principio cada método vectorizador sea más apropiado para un algoritmo concreto, vamos a probar ambos métodos en ambos algoritmos, para comparar su eficacia y corroborar la afirmación anterior, igual que se hace en el artículo [53], donde se compara el rendimiento de distintos algoritmos según el tipo de vectorización.

4.6. Selección de algoritmos de aprendizaje

Para abordar la clasificación de correos electrónicos como Spam o Ham, hemos seleccionado dos técnicas de aprendizaje supervisado bien conocidas por su efectividad en problemas de clasificación de texto.

El clasificador **Naive Bayes Multinomial** es adecuado para la clasificación de textos debido a su simplicidad y eficacia en el manejo de grandes conjuntos de datos con muchas características, como es típico en problemas de Procesamiento del Lenguaje Natural. Este clasificador funciona bien con características discretas, como los recuentos de palabras obtenidos mediante `CountVectorizer`. Aunque podríamos considerar otros clasificadores bayesianos, como el Bernoulli o el Gaussian Naive Bayes (con una preparación de los datos menos convencional), el Multinomial es preferido en este contexto debido a su adecuación con la distribución de los recuentos de palabras.

El clasificador **Naive Bayes de Bernoulli** podría usarse en este contexto si decidieramos transformar el conjunto de datos a un formato binario, donde cada palabra del vocabulario sea representada como una característica que indica su presencia o ausencia (1 o 0) en cada correo. El problema de esto es que al no considerar la frecuencia de cada palabra, perdemos mucho y no podemos hacer selección de características, lo que mermaría la eficiencia en la fase de entrenamiento.

Para **SVM**, inicialmente consideramos utilizar el algoritmo `SVC` para la tarea de clasificación de *spam*, dado su renombre en producir modelos robustos y eficientes que funcionan bien en espacios de alta dimensión. Sin embargo, nos encontramos con limitaciones significativas en términos de eficiencia computacional y tiempo de ejecución cuando se trataba de conjuntos de datos grandes. Según la documentación de `scikit-learn` [48], para datasets grandes se

recomienda utilizar `LinearSVC` o `SGDClassifier`, ya que estos modelos son más eficientes con grandes volúmenes de datos.

En resumen, hemos seleccionado los algoritmos de aprendizaje `MultinomialNB`, `BernoulliNB`, `SVC`, `LinearSVC` y `SGDClassifier`.

4.6.1. Selección de hiperparámetros

Los **hiperparámetros** son aquellos parámetros que se configuran de manera manual antes de entrenar un modelo (por ejemplo, número máximo de iteraciones, número de nodos, tasa de aprendizaje, tipo de kernel, etc). No debemos confundirlos con los parámetros, que son elementos internos derivados de manera automática durante el proceso de aprendizaje y que no están configurados manualmente por científicos de datos.

La **selección de hiperparámetros** consiste en encontrar la combinación de parámetros más óptima para un clasificador. Estos parámetros influyen enormemente en el comportamiento del modelo entrenado. Diferentes valores de estos hiperparámetros pueden dar lugar a modelos muy distintos en términos de eficacia, eficiencia y su capacidad de generalización (su capacidad para clasificar correctamente instancias nuevas).

`Scikit-learn` ofrece varias herramientas robustas para automatizar la selección de hiperparámetros:

- `GridSearchCV`: esta herramienta permite definir un grid o una cuadrícula de hiperparámetros que serán explorados exhaustivamente, realizando pruebas con todas las combinaciones posibles. Además utiliza validación cruzada para evaluar cada combinación, lo cual nos asegura que el rendimiento del modelo sea menos afectado por las peculiaridades que pueda tener una partición particular de los datos.

- `RandomizedSearchCV`: a diferencia de GridSearchCV, esta opción no prueba todas las combinaciones posibles, sino que selecciona al azar un número fijo de combinaciones de hiperparámetros a probar. Esto puede ser más eficiente desde el punto de vista computacional, especialmente cuando se trabaja con un espacio de hiperparámetros grande. Usa también validación cruzada.

En un principio no iba a trabajar con un conjunto de hiperparámetros demasiado grande, por lo que decidí usar `GridSearchCV` para todos los clasificadores, pues explorar todas las combinaciones posibles puede dar mejores resultados. Sin embargo, la selección de hiperparámetros para el algoritmo SVC tiene un alto tiempo de ejecución (entre 2 y 10 minutos por iteración), por lo que para este modelo voy a aplicar `RandomizedSearchCV`.

4.6.1.1. SVM

El clasificador `SVC` se ha configurado utilizando una gran variedad de hiperparámetros. Los hiperparámetros más relevantes y su interpretación son los siguientes:

- **C**: Este hiperparámetro regula la penalización del término de error en la función objetivo del SVM. Como vimos en 3.2.4.2, valores más altos de C resultan en un margen más pequeño y un ajuste más estricto al conjunto de entrenamiento, mientras que valores más bajos permiten un margen más grande y un modelo más generalizado. En nuestro

enfoque, hemos explorado $C = \{0.1, 1, 10\}$ para encontrar el equilibrio óptimo entre sesgo y varianza.

- **gamma**: El parámetro γ define cuánta influencia tiene un solo ejemplo de entrenamiento. Un valor bajo indica que el radio de influencia de los soportes es grande, mientras que un valor alto resulta en radios más pequeños, afectando principalmente los soportes más cercanos. Los valores explorados incluyen "scale", "auto", 0.1, 1, 10, donde "scale" y "auto" ajustan automáticamente γ basándose en las propiedades del conjunto de datos.

- **kernel**: El tipo de función del kernel utilizado para transformar el espacio del problema original a un espacio dimensional más alto donde los datos pueden ser linealmente separables. Hemos considerado los siguientes tipos de kernel:
  - `rbf`: kernel de base radial, útil para espacios de entrada de tamaño moderado.
  - `linear`: kernel lineal, adecuado para grandes espacios de características.
  - `poly`: kernel polinómico, que introduce términos polinómicos en la función de decisión.

Para el clasificador **LinearSVC**, utilizamos un enfoque de márgenes lineales sin la transformación de kernel. Hemos configurado los hiperparámetros de la siguiente manera:

- **C**: Al igual que en SVC, ajusta la penalización de los términos erróneos. Probamos con $C = \{0.1, 1, 10\}$.

- **max_iter**: Define el número máximo de iteraciones que el algoritmo realiza durante la optimización. Exploramos valores de $\{1000, 5000, 10000\}$ para garantizar la convergencia.

Por último, el clasificador **SGDClassifier** implementa modelos lineales con aprendizaje estocástico por descenso de gradiente, y se han configurado los siguientes hiperparámetros:

- **alpha**: Equivalente a la inversa de la regularización C en otros modelos lineales, con valores explorados de $\{0.0001, 0.001, 0.01\}$.

- **loss**: Define la función de pérdida a utilizar. Consideramos opciones como 'hinge' para un equivalente a SVM, 'log' para regresión logística, y 'modified_huber' para una versión robusta de la pérdida de bisagra.

- **penalty**: Establece el tipo de regularización: 'l2', 'l1', o 'elasticnet', cada una afectando de manera diferente la reducción de la dimensión y la tolerancia al error en el modelo.

4.6.1.2. Naive Bayes

Para Naive Bayes Multinomial y de Bernoulli, sólo vamos a considerar el hiperparámetro `alpha`. Este hiperparámetro controla el suavizado de Laplace, implementado para manejar el problema de la probabilidad cero en el cálculo de probabilidades condicionales. Un valor de `alpha` > 0 ayuda a manejar las características que no aparecen en las muestras de aprendizaje y evita que las probabilidades se multipliquen por cero. En nuestro caso, exploraremos un rango de valores de `alpha` $= \{0, 0.01, 0.1, 1.0\}$ para encontrar el que mejor mejore el rendimiento del modelo resultante.

4.7. Validación y evaluación de clasificadores

La evaluación y la validación de clasificadores son dos conceptos fundamentales en el Aprendizaje Automático, especialmente en la construcción y despliegue de modelos de clasificación. Aunque están estrechamente relacionados y a menudo se usan de manera intercambiable en la práctica, tienen enfoques y objetivos distintos. Veamos las diferencias clave entre ambos [26]:

La **validación** es el proceso de verificar si el modelo de clasificación es capaz de generalizar bien a partir de nuevos datos que no se utilizaron durante el entrenamiento. Su propósito principal es asegurar que el modelo funcione de manera efectiva en distintos conjuntos de datos y minimizar el riesgo de sobreajuste.

La **evaluación** implica medir la eficacia de un modelo de clasificación ya validado y listo para ser desplegado, utilizando métricas específicas sobre un conjunto de datos de prueba. Este conjunto de datos debe ser independiente de los datos utilizados para entrenamiento y validación.

4.7.1. Validación de clasificadores

A la hora de construir clasificadores, es crucial cuantificar y determinar su eficacia. Por ejemplo, en aplicaciones críticas como la detección de enfermedades, no podemos permitir un alto índice de fallos. En la evaluación de un clasificador se deben tener presentes varios criterios: costes computacionales de tiempo, simplicidad, interpretabilidad del modelo obtenido y su precisión (*accuracy*). Esta última es la medida que más atención suele recibir, pues refleja la proporción de predicciones correctas, o el número de aciertos sobre el total de casos (Ver apartado 4.7.2).

Además de usarse para evaluar clasificadores, la precisión también puede ser utilizada durante el propio proceso de construcción del clasificador [29]:

- **Estimación por resustitución (*resubstitution estimate*):** este método utiliza los mismos datos para entrenar y para evaluar el clasificador.
- ***Holdout*:** consiste en dividir los datos en dos conjuntos: uno de entrenamiento, para construir el clasificador, y otro de prueba, para evaluar la precisión del clasificador de forma independiente y así evitar sesgos. Normalmente el conjunto de entrenamiento contiene la mayoría del conjunto de datos (entre un 60 % y un 80 %).
- **Remuestreo (*random subsampling*)**: es similar al método anterior pero se realizan varias particiones de los conjuntos de entrenamiento y prueba. La precisión del clasificador se obtiene como la media para los distintos conjuntos de prueba.
- **Validación cruzada (*cross-validation*):** se realizan k particiones de igual tamaño del conjunto de datos, y a lo largo de k iteraciones se utilizan k-1 particiones para entrenar el clasificador, para luego validar con la partición sobrante. La precisión del clasificador se obtiene como la media de las k precisiones calculadas en cada iteración.
- **Dejar uno fuera (*leave-one-out*)**: se trata de un caso particular de *cross-validation*, donde el número de particiones k es igual al tamaño del conjunto de datos. Este método sólo es óptimo para conjuntos de datos muy pequeños pues tener que construir k clasificadores resulta muy costoso computacionalmente.

- ***Bootstrapping*:** este método se basa en tomar múltiples muestras del conjunto de datos original con reemplazo, construyendo así nuevos conjuntos de datos con el mismo tamaño que el original, donde puede haber datos repetidos. Cada conjunto de datos se usa para entrenar un modelo, y las predicciones de estos modelos se utilizan para obtener una medida más robusta de la precisión o de otras métricas de rendimiento.

Además de los métodos de evaluación anteriores, podemos aplicar combinaciones de estos para un rendimiento del modelo aún mejor. Lo más común es combinar las técnicas de *holdout* y de validación cruzada. Inicialmente se divide el conjunto de datos en entrenamiento y prueba mediante *holdout*. Así se garantiza que el modelo se evalúe con datos que no conoce; de esta forma se evita el sesgo en la evaluación final. Por otro lado, se aplica validación cruzada dentro del conjunto de entrenamiento, lo cual asegura que se maximice el aprendizaje y la generalización, al entrenar el modelo múltiples veces rotando los conjuntos de entrenamiento y prueba. En la figura 4.10 podemos observar el esquema del método de evaluación propuesto, para $k = 5$.

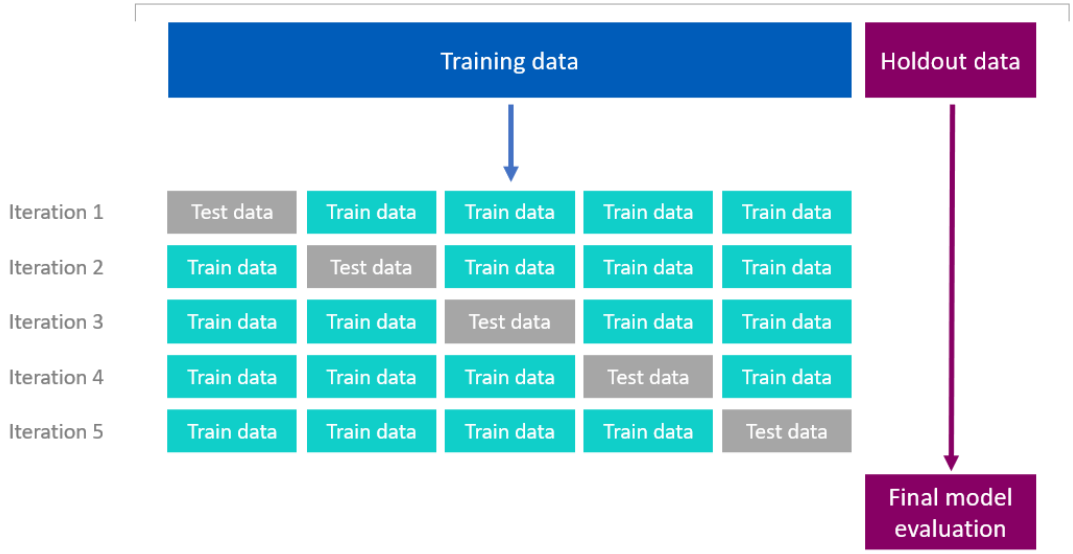


Figura 4.10.: Combinación de holdout y cross-validation [7]

En nuestra implementación, hemos elegido el método de *holdout* para dividir los datos en un conjunto de entrenamiento (80 % de los datos) y otro de test (20 % de los datos), mediante la función `train_test_split` (Apéndice B.2.2). Es importante recalcar que el modelo será entrenado únicamente con ayuda del conjunto de entrenamiento, y no usaremos el conjunto de test hasta el final, para **evaluar** el modelo final.

Para el entrenamiento del modelo hemos elegido el método de **validación cruzada estratificada**, proporcionado por la función `StratifedKFold` (Apéndice B.2.2). Este método tiene de particular que al dividir el conjunto de datos, mantiene la proporción de tamaños de cada clase. Por ejemplo, si la proporción de instancias Spam es del 40 % en nuestro conjunto de datos, y por tanto la de Ham es del 60 %, al dividir el conjunto de datos mantenemos esta proporción en las k divisiones. Esto se hace para asegurar que el modelo sea evaluado de

manera justa en todas las divisiones, evitando sesgos que podrían surgir si alguna división resultara desproporcionada.

4.7.2. Métricas de evaluación

Tomando como positiva la clase Spam y como negativa la clase Ham, vamos a definir varias de las métricas que nos servirán para evaluar el rendimiento y la eficacia del modelo construido [3] [54].

En primer lugar presentamos la **matriz de confusión** (tabla 4.3), que nos proporcionará una visión general del rendimiento del modelo, al mostrar cuántas instancias se han clasificado correctamente e incorrectamente en cada clase. La matriz se compone de las siguientes cuatro medidas, definidas en el contexto de nuestro problema de la siguiente forma:

- **TP (verdaderos positivos):** el número de instancias Spam que han sido correctamente etiquetadas como Spam.
- **FP (falsos positivos):** el número de instancias Ham que han sido incorrectamente etiquetadas como Spam.
- **TN (verdaderos negativos):** el número de instancias Ham que han sido correctamente etiquetadas como Ham.
- **FN (falsos negativos):** el número de instancias Spam que han sido incorrectamente etiquetadas como Ham.

		Clase real	
		Positiva	Negativa
Clase predicha	Positiva	Verdaderos Positivos (TP)	Falsos Positivos (FP)
	Negativa	Falsos Negativos (FN)	Verdaderos Negativos (TN)

Tabla 4.3.: Matriz de confusión para un problema con dos clases

A partir de esta matriz, podemos calcular diversas métricas de evaluación del modelo [3], que las definimos ya en el contexto de nuestro problema:

- **Sensibilidad (*recall*):** También conocida como *Tasa de verdaderos positivos (TPR)*, es utilizada para conocer la proporción de instancias Spam que son etiquetadas correctamente.

$$Sensibilidad = \frac{TP}{TP + FN}$$

- **Especificidad (*specificity*):** También conocida como *Tasa de verdaderos negativos (TNR)*, es utilizada para conocer la proporción de instancias Ham que son etiquetadas correctamente.

$$Especificidad = \frac{TN}{TN + FP}$$

- **Precisión (*accuracy*):** Es la proporción de instancias (tanto Spam como Ham) que han sido etiquetadas correctamente.

$$Precisión = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Exactitud (*precision*):** Es la proporción de instancias Spam bien clasificadas entre todos los casos clasificados como positivos.

$$Exactitud = \frac{TP}{TP + FP}$$

- **Tasa de error (*error rate*):** Es la proporción de instancias (tanto Spam como Ham) que han sido etiquetadas incorrectamente.

$$ERR = \frac{FP + FN}{TP + TN + FP + FN}$$

- **F1-Score:** es una métrica muy utilizadas en problemas desbalanceados. Se construye a partir de la media armónica de la exactitud y la sensibilidad (recall).

$$F1\text{-}score = 2 \cdot \frac{exactitud \cdot recall}{exactitud + recall}$$

La **curva ROC** (*Receiver Operating Characteristic*) se trata de una estrategia popular para evaluar el rendimiento de los clasificadores en problemas binarios [3] [8]. Gracias a esta, podemos entender la capacidad de discriminación de un clasificador.

Visualmente, la curva ROC representa la tasa de verdaderos positivos (sensibilidad) en el eje Y, frente a la tasa de falsos positivos ($1-$especificidad) en el eje X, del clasificador. La curva expresa cómo varía TPR conforme aumentamos FPR. Esta curva comienza en el punto $(0,0)$, cuando todos los puntos se clasifican como negativos, y acaba en el $(1,1)$, cuando todos los puntos se clasifican como positivos.

En la figura 4.11 podemos observar las curvas ROC para tres clasificadores hipotéticos distintos:

- Un **clasificador *random* o aleatorio** se caracteriza porque su curva ROC es una línea diagonal, donde el número de verdaderos positivos es siempre aproximadamente igual al de falsos positivos. Dicho clasificador no sigue ningún criterio y por tanto su poder predictivo es nulo.
  Este clasificador sirve como un punto de referencia para evaluar otros modelos de clasificación. Cualquier modelo decente debe superar el rendimiento de un clasificador aleatorio. Si un modelo no puede superar a un clasificador aleatorio, entonces es incapaz de aprender ningún patrón útil sobre los datos.

- Para un **clasificador ajustado** a los datos del problema, su curva ROC estará por encima de la línea diagonal. Muestra el rendimiento que un clasificador real puede desempeñar.
- Un **clasificador perfecto o ideal** se representaría como un punto en la esquina superior izquierda del gráfico ROC, correspondiente a un FPR de 0 y un TPR de 1, indicando que no hay falsos positivos y todos los verdaderos positivos son identificados correctamente.

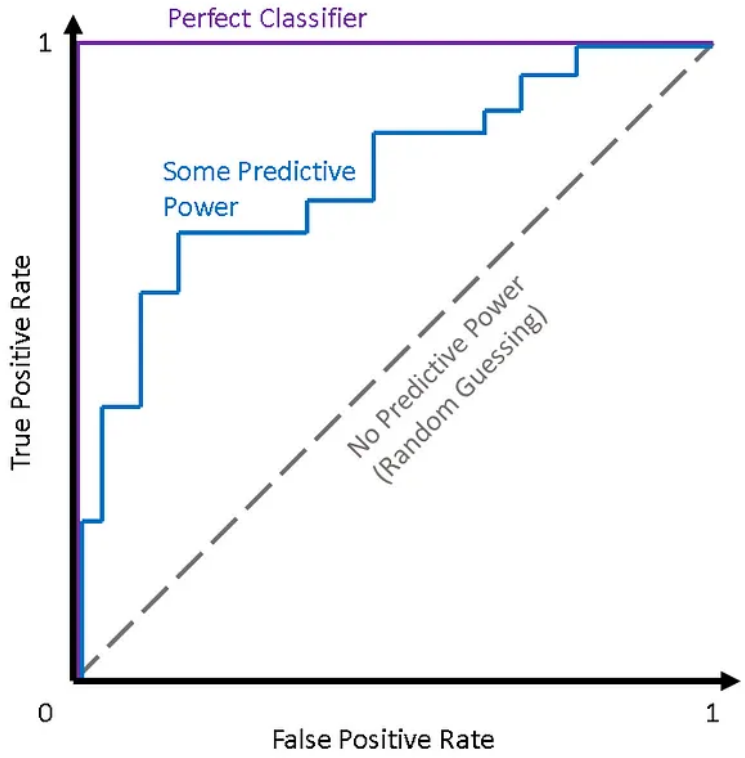


Figura 4.11.: Curva ROC [8]

Aunque puede resultar útil visualizar la curva ROC de un clasificador, existe una métrica que nos aporta esta información. El **área bajo la curva ROC (AUC)** permite representar en un único valor el rendimiento del clasificador. Esto puede resultar útil para realizar comparativas entre clasificadores.

Para clasificadores sin poder predictivo (por ejemplo, el aleatorio), AUC= 0.5, mientras que para un clasificador perfecto, AUC= 1.0. La mayoría de clasificadores tienen un valor de AUC entre 0.5 y 1.0.

Una ventaja de las curvas ROC es que nos permiten ajustar los criterios de decisión de nuestros modelos, para que se alineen mejor con los requisitos del problema en cuestión.

Por ejemplo, para un clasificador de correos electrónicos no deseados, es crucial mantener una tasa de falsos positivos muy baja para evitar que correos importantes sean erróneamente marcados como *spam*, lo cual podría suceder si el algoritmo fuese excesivamente riguroso.

Por lo tanto, estaríamos dispuestos a tolerar que una cantidad considerable de *spam* real (verdaderos positivos) no sea filtrada con tal de minimizar el riesgo de perder erróneamente correos legítimos.

4.8. Resultados y comparación entre los modelos entrenados

Los resultados mostrados en esta sección han sido extraídos del cuaderno `Clasificacion.ipynb`.

4.8.1. Métricas obtenidas

En las tablas 4.4 y 4.5 podemos encontrar las métricas obtenidas por cada clasificador, para fase de entrenamiento y fase de prueba, respectivamente.

Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial (Count)	0.980075	0.980180	0.980056	0.980073	0.996390
NB Multinomial (TF-IDF)	0.991315	0.991322	0.991311	0.991315	0.999441
NB Bernoulli (Count)	0.951465	0.955146	0.951338	0.951356	0.997909
SVC (Count)	0.999546	0.999546	0.999546	0.999546	1.000000
SVC (TF-IDF)	0.999886	0.999887	0.999886	0.999886	0.999990
LinearSVC (Count)	0.999205	0.999206	0.999205	0.999205	0.999995
LinearSVC (TF-IDF)	0.999319	0.999318	0.999319	0.999319	0.999842
SGDClassifier (Count)	0.996821	0.996829	0.996817	0.996821	0.999848
SGDClassifier (TF-IDF)	0.999092	0.999091	0.999092	0.999092	0.999772

Tabla 4.4.: Métricas en fase de entrenamiento

Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial (Count)	0.973660	0.973650	0.973708	0.973659	0.992914
NB Multinomial (TF-IDF)	0.981153	0.981137	0.981179	0.981152	0.998545
NB Bernoulli (Count)	0.944369	0.947014	0.944828	0.944319	0.994564
SVC (Count)	0.980018	0.980116	0.980134	0.980018	0.996037
SVC (TF-IDF)	0.992961	0.992953	0.992968	0.992960	0.999390
LinearSVC (Count)	0.986376	0.986399	0.986457	0.986376	0.997730
LinearSVC (TF-IDF)	0.990463	0.990466	0.990529	0.990463	0.999343
SGDClassifier (Count)	0.988874	0.988879	0.988942	0.988873	0.997524
SGDClassifier (TF-IDF)	0.990463	0.990461	0.990524	0.990463	0.999277

Tabla 4.5.: Métricas en fase de prueba

En general, se han obtenido resultados muy prometedores para cada representación, lo cual podemos ver con más detalle en las figuras 4.12 y 4.13, donde representamos un gráfico de barras con el valor de *accuracy* dependiendo del modelo y su representación. Este gráfico ayuda a visualizar directamente las diferencias en el rendimiento de los clasificadores bajo diferentes configuraciones vectoriales.

El *accuracy*, que medía la cantidad de aciertos con respecto al total, consigue un valor muy cercano a 1 en todos los casos en entrenamiento, lo cual indica que el número de instancias predichas incorrectamente es casi residual. En la fase de prueba, los valores decrecen un

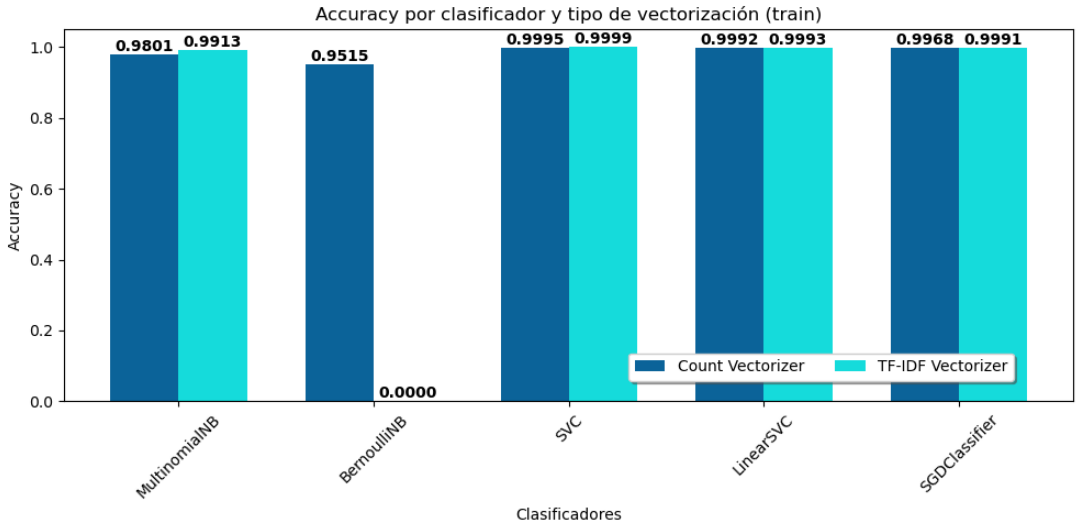


Figura 4.12.: Combinación de valores de *accuracy* en *train*

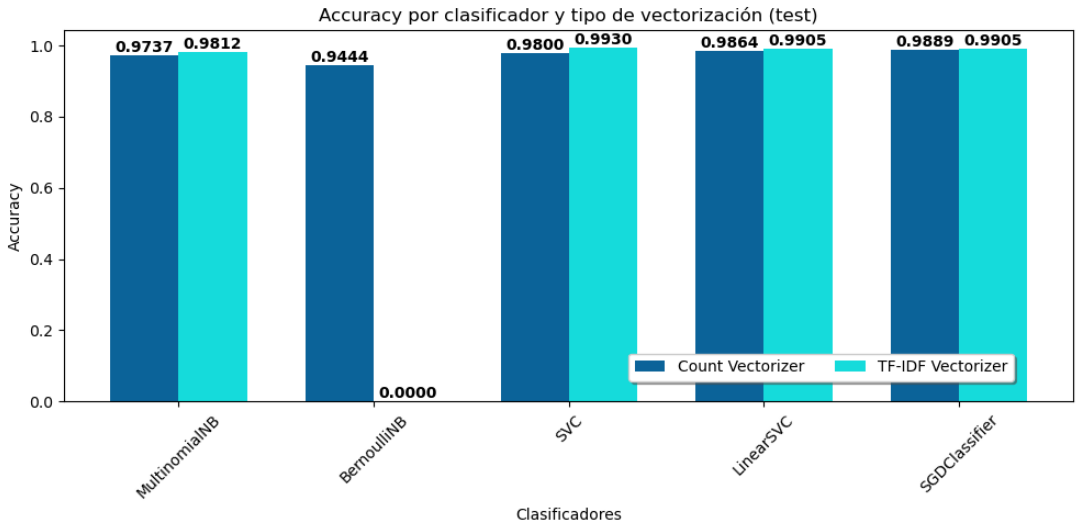


Figura 4.13.: Combinación de valores de *accuracy* en *test*

poco, como es típico, pero siguen siendo muy competentes. Por tanto podemos concluir que la capacidad de los modelos para generalizar es bastante buena.

Con respecto a las otras métricas como *precision*, la cual medía el número de aciertos con respecto a las instancias clasificadas como positivas, se consiguen resultados en consonancia con las demás métricas, obteniendo valores muy altos. Esto indica que apenas se confunden correos Ham como Spam.

En cuanto al *recall*, que mide el *TPR*, también es muy alto. Esto significa que de entre los positivos (Spam), clasifica correctamente a prácticamente todos ellos, con lo cual no se

confunden correos Spam como Ham.

Además, el *F1-Score* reafirma los resultados de las dos métricas anteriores, pues al estar calculada a partir de estas, obviamente obtiene valores elevados. Esto indica que se consigue un equilibrio entre *precision* y *recall*.

Observamos que los resultados obtenidos son casi perfectos, excepto en el caso de Naive Bayes de Bernoulli, donde tiene valores de las métricas ligeramente más bajos. Esto puede deberse a que al tener sólo en cuenta la presencia o ausencia de cada término en las instancias, esto nos hace perder mucha información.

En la figura 4.14 podemos visualizar las **matrices de confusión** para Naive Bayes Multinomial y de Bernoulli, tanto para entrenamiento como para fase de prueba. Esta nos permite observar la capacidad de cada modelo para clasificar correctamente las instancias de las clases. En entrenamiento, Naive Bayes Multinomial clasificó correctamente 8717 instancias como Spam y 8548 como Ham, mientras que clasificó erróneamente 235 como Ham y 116 como Spam. En la fase de prueba obtuvo resultados proporcionales, y esta es la razón por la que las métricas son tan buenas (apenas tiene falsos positivos y falsos negativos).

En cambio, para Naive Bayes Bernoulli, que es el modelo con peor desempeño, la cosa cambia un poco. A pesar de que el número de falsos positivos es muy pequeño (incluso más pequeño que el caso Multinomial), el problema lo tenemos en los falsos negativos. Obtenemos unas cifras de 818 falsos positivos en entrenamiento y 213 en prueba, que prácticamente cuadruplican a los obtenidos en el caso Multinomial. Esto se traduce en que nuestro modelo de Bernoulli no es capaz de mantener la tasa de falsos positivos muy baja, lo cual indicábamos en 4.7.2 que era crucial para que el clasificador no coloque correos legítimos en la bandeja de *spam* y haga perder al usuario información importante. Por esta razón, Naive Bayes Bernoulli sería un mal modelo con respecto a los demás, al permitir que esto ocurra.

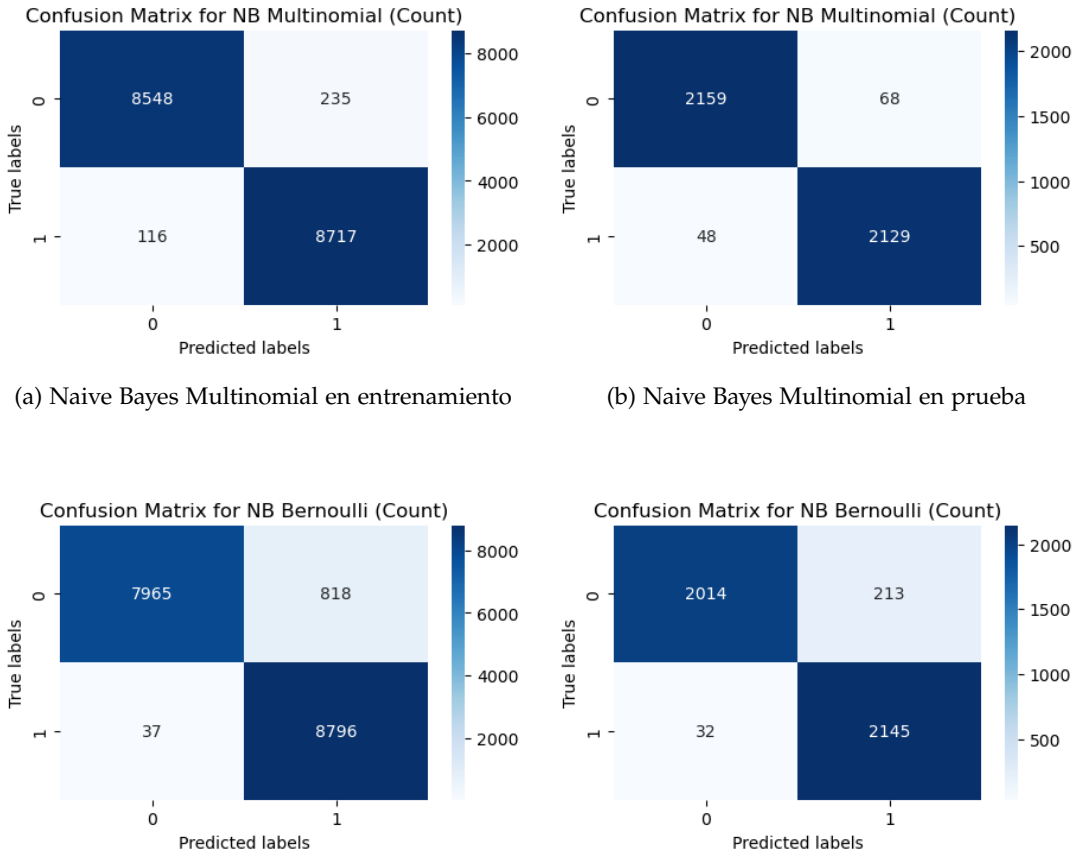
La métrica *AUC* también demuestra gran consistencia, la cual indica que el clasificador es capaz de discernir entre clases positivas y negativas. Esto lo podemos confirmar observando las curvas ROC de la figura 4.15. Observamos que todas las áreas bajo la curva son casi el total, incluso para SVC, a pesar de que hemos realizado muy pocas iteraciones en la búsqueda de hiperparámetros. Esto parece deberse a que los algoritmos seleccionados son altamente efectivos en configuraciones por defecto para este tipo de problemas, y y son capaces de rendir bien incluso con limitaciones. Adicionalmente, la preparación previa de los datos puede haber contribuido a que tengamos un espacio de características separable, lo que reduce la necesidad de ajustes complejos en los hiperparámetros.

En general, los modelos han demostrado un rendimiento prácticamente inmejorable, resaltando la robustez de los algoritmos basados en SVM, especialmente con el uso de la vectorización TF-IDF, que parece mejorar un poco la capacidad del modelo para diferenciar entre las clases, con respecto a vectorización normal.

Estos resultados demuestran la efectividad de las técnicas avanzadas de vectorización y clasificación empleadas en la tarea de filtrado de *spam*. Además, sugiere que las técnicas de procesamiento elegidas han impactado de forma positiva a la calidad de la clasificación.

4.8.2. Top palabras predictoras

A partir de los modelos de clasificación basados en Máquinas de Soporte Vectorial, podemos extraer los pesos asignados a las características o términos. El objetivo es identificar las



(a) Naive Bayes Multinomial en entrenamiento

(b) Naive Bayes Multinomial en prueba

(c) Naive Bayes Bernoulli en entrenamiento

(d) Naive Bayes Bernoulli en prueba

Figura 4.14.: Comparación de matrices de confusión

palabras que tienen mayor influencia en la clasificación de los mensajes como Spam o Ham.

Esto solo lo podemos aplicar cuando utilizamos un kernel lineal, como en `LinearSVC` o `SVC(kernel='linear')`. El atributo `coef_` representa los coeficientes del hiperplano usado para separar las clases. Para modelos multiclase, `coef_` puede contener varias filas, una por clase. El atributo `coef_[0]` representa los coeficientes del hiperplano de decisión para la primera clase contra todas las otras clases. Como nuestro caso es binario, sólo necesitamos extraer `coef_[0]`.

El atributo `coef_` devuelve una matriz que puede ser dispersa, por lo que tenemos que transformarla a densa. Mediante la función `argsort()`, se ordenan los índices de los pesos en orden ascendente. Para los predictores de Ham, se seleccionan los primeros 15 índices, correspondientes a los pesos más negativos. En cambio, para los predictores de *spam*, se seleccionan los últimos 15 índices, que corresponden a los pesos más positivos. A continuación, a partir del vectorizador se toman los términos asociados a dichos índices.

En la figura 4.16 podemos observar los 15 términos más relevantes a la hora de clasificar en Ham o Spam. Esto significa que si aparece dicha palabra en un correo, el modelo lo clasificará muy seguramente como Ham o Spam respectivamente.

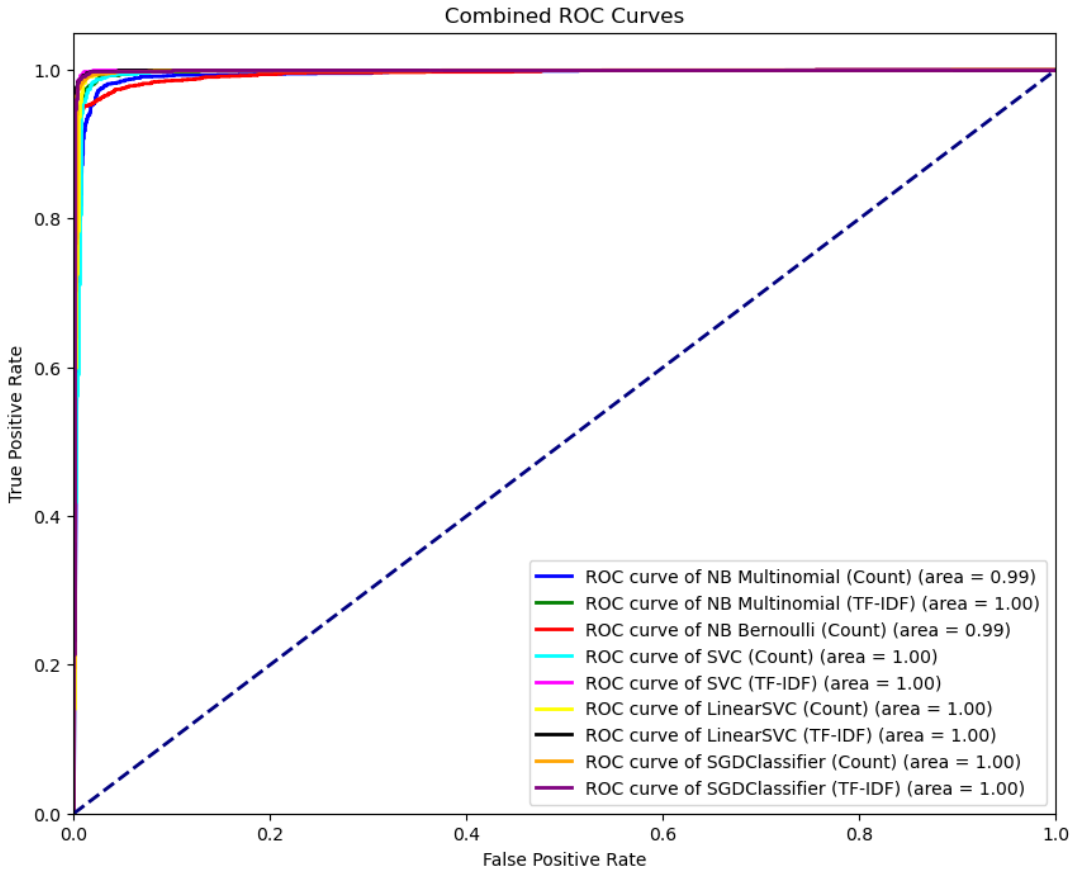


Figura 4.15.: Comparación de curvas ROC

```
Top predictors of Spam:
click        (0.931283)
instructions (0.795358)
wish         (0.745940)
xm           (0.720586)
sightings    (0.681662)
arabia       (0.652055)
saudi        (0.627169)
knew         (0.606327)
dating       (0.605892)
tel          (0.605512)
cards        (0.574324)
refuse       (0.552601)
carry        (0.552600)
gpo          (0.552598)
merchant     (0.552597)
```

```
Top predictors of Ham:
enron      (-1.291580)
date       (-1.111402)
revised    (-1.000114)
avi        (-1.000000)
hart       (-0.997911)
tw         (-0.913973)
vince      (-0.887259)
california (-0.850464)
thanks     (-0.825936)
gif        (-0.818813)
wrote      (-0.810769)
deserved   (-0.786866)
louise     (-0.786121)
serv       (-0.771182)
attached   (-0.761796)
```

Figura 4.16.: Top palabras predictoras de Spam y Ham

De entre los términos identificados como principales predictores de correos *spam* destacan 'click', 'dating' y 'cards', que pueden estar relacionados con incitar a los usuarios a seguir instrucciones para reclamar supuestas recompensas, completar transacciones o registrarse en servicios de citas en línea.

Por otro lado, en los predictores de correos *ham*, términos como 'enron', 'attached' y 'revised', están más relacionados con el ambiente laboral y corporativo, por lo que pueden ser indicativos de una comunicación legítima dentro de una empresa.

Estos predictores pueden usarse para entender mejor los modelos generados, pues a diferencia de otros enfoques como Árboles de decisión, nuestros modelos no generan reglas ni nada que consiga explicar por qué toman una decisión u otra. Por tanto esta información nos ayuda a entender cómo el modelo toma decisiones basadas en el texto de los mensajes y también identificar posibles sesgos en las características aprendidas.

4.9. Otras representaciones: Synsets

Además del uso de tokens para representar los términos de cada documento, hemos querido probar con nuevos enfoques en el ámbito del Procesamiento del Lenguaje Natural. En particular nos hemos inspirado en la tesis [55], donde se emplea el uso de *synsets*, que identifican grupos de palabras sinónimas haciendo uso de **diccionarios ontológicos** como *WordNet* [56].

Un *synset*, o conjunto de sinónimos, agrupa diferentes palabras que expresan una idea similar. Este enfoque permite modelar con más precisión las relaciones semánticas entre términos y mejorar la comprensión del contexto dentro de los documentos.

El procedimiento que hemos seguido para transformar nuestro conjunto de datos es el siguiente:

1. **Lematización**: Utilizamos la función `lemmatize()` de la biblioteca `NLTK` para transformar cada palabra a su forma base. Este paso es necesario para garantizar que algunas palabras, como tiempos verbales o formas plurales, sean consolidadas en una única representación antes de buscar sus *synsets* en `WordNet`.

2. **Identificación de *synsets*:** Para cada palabra en un documento, identificamos su conjunto de sinónimos (*synset*) más representativo, utilizando la función `synsets()` de `WordNet`.

3. **Extracción de hiperónimos:** A partir del *synset* obtenido, buscamos su primer hiperónimo, mediante la función `hypernyms()` de `WordNet`. Un hiperónimo es un término más general que describe una categoría o clase a la que la palabra original pertenece.

4. **Reemplazo de términos:** Si un hiperónimo está disponible, reemplazamos la palabra original por este término más general en el texto. Si no hay hiperónimos disponibles, mantenemos la palabra original. Este enfoque nos permite abstraer las palabras a un nivel superior, capturando un significado más general y reduciendo la ambigüedad.

Esta técnica de transformación se aplica sistemáticamente a todas las palabras de cada correo, lo que produce una versión transformada del texto donde cada término está alineado con un concepto más general (su hiperónimo). De esta forma, tratamos de enriquecer el modelo de clasificación, mediante la detección palabras que tienen una relación semántica.

Por ejemplo, según esta técnica, los términos 'dollar' y 'euro' se transforman ambos en 'monetary_unit', pasando de ser términos sin relación alguna a representar lo mismo.

En la figura 4.6 podemos ver un ejemplo de cómo quedan algunos correos al procesar sus términos de dicha forma. Observamos que hay palabras, como 'home', que se transforma en el hiperónimo 'residence'. En este caso, modelo podría ser capaz de manejar variaciones léxicas como 'house', 'apartment', etc., lo cual nos permite manejar más sencillamente el texto.

Sin embargo, esta técnica no es perfecta, pues existen palabras que son transformadas suponiendo un contexto erróneo, como 'fight', que se refiere a combatir una enfermedad, y se transforma en 'military_action', que nada tiene que ver.

Label	**Texto limpio**	**Texto tokenizado**
Spam	fight risk cancer URL slim guaranteed lose lbs days URL	['military_action', 'danger', 'malignant_tumor', 'address', 'change_state', 'pledge', 'lose', 'avoirdupois_unit', 'time_unit', 'address']
Spam	make week using home computer free software computer start making huge amounts cash working	['kind', 'time_period', 'mistreatment', 'residence', 'machine', 'people', 'code', 'machine', 'beginning', 'production', 'huge', 'assets', 'currency', 'excavation']
Ham	intern compensation sally summer analysts benefit recent compensation changes monthly salary summer let know questions	['doctor', 'recompense', 'remark', 'season', 'expert', 'payment', 'recent', 'recompense', 'happening', 'series', 'regular_payment', 'season', 'let', 'knowing', 'questioning']

Tabla 4.6.: Comparación de textos limpios y tokenizados a partir de *synsets*

El uso de *synsets* ofrece varias ventajas para el análisis semántico, pues permite generalizar conceptos y reducir la dimensionalidad del espacio de características, al tratar múltiples palabras similares como una única entidad.

En vista de los resultados obtenidos en el punto anterior, para el conjunto de datos formado por tres fuentes de *spam* distintas, hemos decidido implementar esta nueva representación para cada conjunto de datos por separado. Esto es porque hemos tenido anteriormente problemas con algoritmos como `SVC`, pues al tener un conjunto de datos tan grande, el coste computacional era muy elevado (aun así obtuvimos buenos resultados).

Además, vamos a elegir la vectorización TF-IDF, pues vimos que en general produce mejores resultados que la vectorización por frecuencias.

Los resultados obtenidos se encuentran en los notebooks `Clasificacion_EnronSpam.ipynb`, `Clasificacion_LingSpam.ipynb` y `Clasificacion_SpamAssassin.ipynb`.

En las tablas 4.7, 4.8, 4.9, 4.10, 4.11 y 4.12 podemos observar las métricas para los modelos Naive Bayes Multinomial y SVC entrenados con los conjuntos de datos *EnronSpam*, *LingSpam* y *SpamAssassin*. Primero mostramos los resultados obtenidos por los datos en forma de tokens, y seguidamente la comparación con los modelos entrenados con datos en forma de

synsets.

Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial	0.993277	0.993285	0.993275	0.993277	0.999623
NB Multinomial Synsets	0.988128	0.988082	0.988193	0.988125	0.999287
SVC	0.998351	0.998352	0.998350	0.998351	0.999979
SVC Synsets	0.999871	0.999868	0.999873	0.999871	0.999962

Tabla 4.7.: Métricas en fase de entrenamiento (EnronSpam)

Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial	0.978184	0.978187	0.978192	0.978184	0.998335
NB Multinomial Synsets	0.977812	0.977805	0.977844	0.977811	0.997842
SVC	0.982750	0.982783	0.982768	0.982750	0.997807
SVC Synsets	0.980392	0.980541	0.980496	0.980392	0.998129

Tabla 4.8.: Métricas en fase de prueba (EnronSpam)

Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial	1.000	1.000	1.000	1.000	1.000
NB Multinomial Synsets	1.000	1.000	1.000	1.000	1.000
SVC	1.000	1.000	1.000	1.000	1.000
SVC Synsets	1.000	1.000	1.000	1.000	1.000

Tabla 4.9.: Métricas en fase de entrenamiento (LingSpam)

Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial	0.998848	0.998884	0.998812	0.998847	0.999920
NB Multinomial Synsets	0.990366	0.994331	0.969880	0.981621	0.996905
SVC	1.000000	1.000000	1.000000	1.000000	1.000000
SVC Synsets	0.992293	0.995455	0.975904	0.985371	0.999254

Tabla 4.10.: Métricas en fase de prueba (LingSpam)

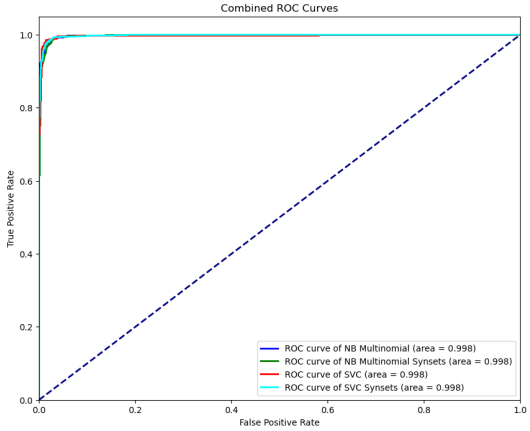
Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial	0.995370	0.995399	0.995349	0.995370	0.999876
NB Multinomial Synsets	0.989133	0.990859	0.980580	0.985581	0.999480
SVC	0.997765	0.997786	0.997749	0.997765	0.999862
SVC Synsets	0.998819	0.999208	0.997692	0.998447	0.999858

Tabla 4.11.: Métricas en fase de entrenamiento (SpamAssassin)

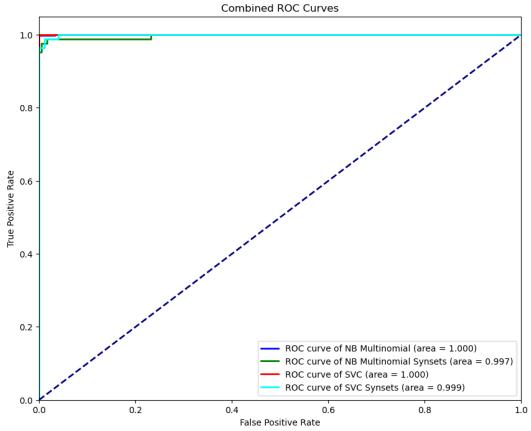
Clasificador	Accuracy	Precision	Recall	F1-Score	AUC
NB Multinomial	0.990421	0.990341	0.990482	0.990408	0.999061
NB Multinomial Synsets	0.970727	0.970919	0.955656	0.962917	0.994958
SVC	0.993614	0.993604	0.993604	0.993604	0.998941
SVC Synsets	0.983947	0.982477	0.977371	0.979883	0.998355

Tabla 4.12.: Métricas en fase de prueba (SpamAssassin)

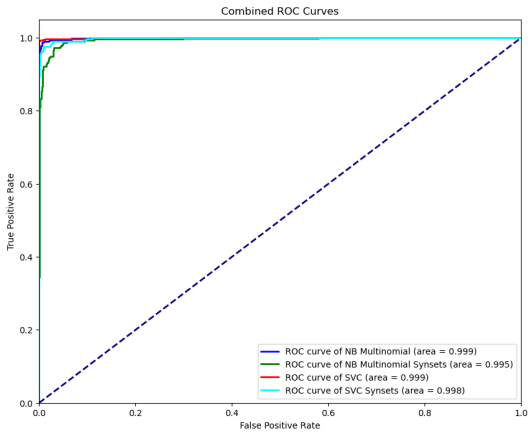
Observamos, al igual que antes, que las métricas son casi perfectas. Además, en la figura 4.17 tenemos la curva ROC para estos modelos, en los distintos conjuntos de datos. Todas las curvas alcanzan otra vez casi la perfección, por lo que concluimos que no hay ningún conjunto de datos problemático.



(a) Curvas ROC para EnronSpam



(b) Curvas ROC para LingSpam



(c) Curvas ROC para SpamAssassin

Figura 4.17.: Comparación de curvas ROC para cada conjunto de datos

Podemos concluir entonces que el uso de *synsets* puede ser útil para reducir el espacio de características relacionando términos entre sí que tienen una conexión semántica. En general, para los conjuntos de datos *EnronSpam* y *LingSpam*, los resultados son prácticamente idénticos a cuando usábamos tokens normales. También es difícil obtener mejores resultados pues estos ya alcanzaban valores muy altos. Sin embargo, en *SpamAssassin*, observamos una pequeña desmejora en las métricas al usar *synsets*. No obstante, esta diferencia es muy pequeña y los resultados siguen siendo notablemente buenos.

5. Conclusiones y trabajos futuros

Uno de los principales objetivos de este trabajo ha sido el de confirmar la viabilidad de detectar correo electrónico basura o *spam* mediante el uso de técnicas convencionales de Minería de Textos. Para ello, se ha utilizado un corpus textual de *Kaggle* y se ha confirmado la existencia de características y patrones distintivos que diferencian los textos *spam* de los legítimos. Por ejemplo, hemos visto que para cada categoría, el contenido de los textos es distinto. Los correos *spam* suelen contener palabras relacionadas con el marketing, las citas y las transacciones, mientras que en los correos *ham* aparecen términos más corporativos y académicos. No obstante, se ha descubierto que características que inicialmente considerábamos determinantes, como la longitud de los textos o la presencia de URLs, no han proporcionado información signficiativa a la hora de distinguir los tipos de correo.

Gran parte de los conocimientos aplicados en este trabajo provienen de diversas asignaturas cursadas en el doble grado, desde asignaturas de Matemáticas como Estadística Descriptiva e Introducción a la Probabilidad, Probabilidad y pequeñas nociones de geometría y análisis, hasta asignaturas de Ingeniería Informática como Aprendizaje Automático, Inteligencia de Negocio e Ingeniería de Conocimiento. Estas disciplinas han contribuido a la base teórica y práctica necesaria para la realización de este trabajo.

Algunos de los problemas y dificultades que se han presentado durante la realización del trabajo son los siguientes: en primer lugar, como hemos mencionado a lo largo de la memoria, la extracción conocimiento a partir de texto en lenguaje natural no es un tarea trivial. El texto no estructurado requiere de un tratamiento más intensivo que los datos numéricos con los que estamos acostumbramos a tratar. Por ello hemos experimentado un proceso de prueba y error donde a veces incluir determinado ajuste en el preprocesamiento podía mermar la eficacia de los modelos entrenados. En segundo lugar, hemos enfrentado problemas con los tiempos de ejecución al entrenar algunos de los algoritmos seleccionados. Esto se ha debido principalmente al volumen del corpus textual y a la representación de los textos mediante frecuencias de términos, lo cual generó un espacio de características muy amplio y ralentizó considerablemente la fase experimental.

En cuanto a los resultados de la clasificación, destacamos el sobresaliente desempeño de los algoritmos empleados. En general se han obtenido métricas de evaluación con valores por encima del 95 %, lo que indica una alta eficacia en la detección de *spam*. Sin embargo, debido a estos altos valores, resulta complicado hacer comparaciones detalladas entre los modelos, ya que todos exhiben un rendimiento casi inmejorable.

A pesar de esto, hemos observado variaciones significativas en la eficacia al cambiar la representación de los datos. La vectorización mediante la medida TF-IDF se ha posicionado como la más eficaz, al conseguir siempre los mejores resultados, en términos de precisión y capacidad de generalización. Esta técnica, al ponderar la importancia de las palabras dentro del documento en relación con su frecuencia en todo el corpus, ayuda a mejorar la discriminación entre las dos clases.

Por otro lado, el uso de técnicas de PLN como el agrupamiento de términos según *synsets* no ha demostrado mejorar los resultados. De hecho, la introducción de este método ha llevado a una ligera disminución en el desempeño de los modelos. Esto podría deberse a que la simplificación semántica puede perder información crucial que es específica del contexto del *spam* o del *ham*, diluyendo así las diferencias clave entre estas categorías.

Sobre posibles ampliaciones y trabajos futuros, en primer lugar podríamos expandir la fase de análisis exploratorio para incluir un análisis de sentimientos que explore las distintas empresiones que transmiten los correos basura con respecto a los legítimos. Este análisis podría revelar tendencias emocionales en el lenguaje que diferencian los dos tipos de correos, como un uso intensivo de palabras que incitan al miedo o a la urgencia en los correos *spam*.

Además, un análisis más detallado del contenido de los enlaces o URLs podría ayudar a detectar intentos de phishing, especialmente a través del análisis del uso de protocolos de seguridad como HTTPs.

Por último, en la fase de clasificación, podríamos emplear algoritmos que resulten en modelos más explicativos, como árboles de decisión, que además de clasificar, proporcionen una comprensión clara de las decisiones tomadas, facilitando así la interpretación de los resultados y la identificación de características clave. También podemos emplear técnicas de Aprendizaje no supervisado, como reglas de asociación, que podrían ofrecer nuevas perspectivas en la detección de *spam*, al permitir descubrir relaciones interesantes entre términos.

A. Planificación y estimación de costes

En la figura A.1 mostramos el diagrama de Gantt con la planificación seguida a lo largo del proyecto. Se comenzó realizando una investigación y búsqueda bibliográfica sobre el tema y las bases matemáticas, se continuó con la redacción de la memoria y, en paralelo, con el desarrollo de la experimentación. Por último, se realizaron revisiones periódicas.

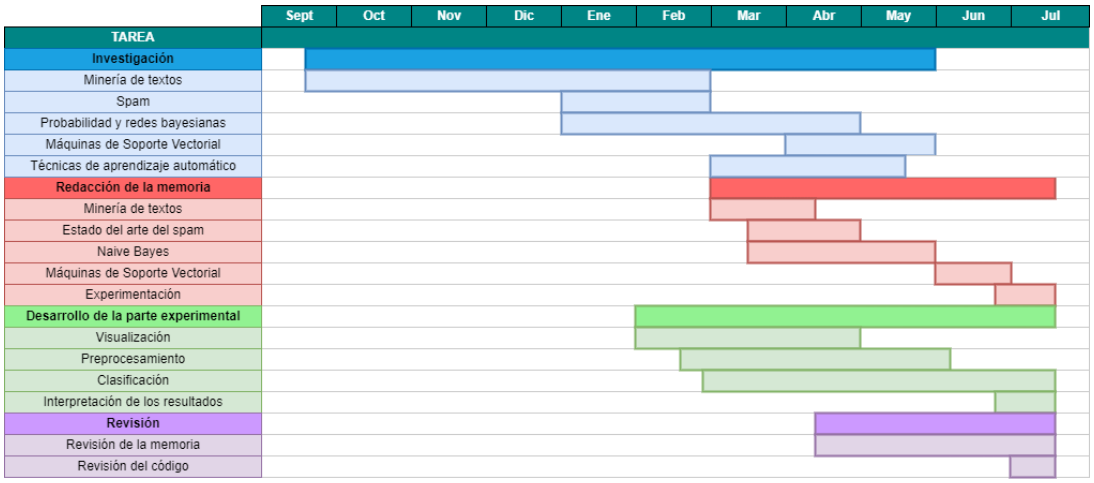


Figura A.1.: Diagrama de Gantt con la planificación del proyecto

Si además queremos calcular un presupuesto por el trabajo realizado, podemos realizar una aproximación de los costes asociados al proyecto. Hemos supuesto que el coste de una hora de trabajo es de 25€, y considerado que la duración del proyecto es de 9 meses. En la tabla A.1 podemos observar la tabla con el cálculo total de los costes.

Tipo	**Tareas**	**Cantidad (h)**	**Coste (€)**
Investigación	Minería de Textos	50	1250
	Spam	15	375
	Probabilidad y redes bayesianas	20	500
	Máquinas de Soporte Vectorial	10	250
	Técnicas de Aprendizaje Automático	50	1250
Redacción de la memoria	Minería de Textos	35	875
	Estado del arte del spam	16	400
	Naive Bayes	45	1125
	Máquinas de Soporte Vectorial	35	875
	Experimentación	42	1050
Experimentación	Visualización	28	700
	Preprocesamiento	25	625
	Clasificación	26	650
	Interpretación de los resultados	15	375
Revisión	Revisión de la memoria	30	750
	Revisión del código	20	500
Total		450	11250

Tabla A.1.: Tabla de costes

B. Software

B.1. Estructura del proyecto

Podemos encontrar el proyecto en la plataforma de desarrollo *Github*, en https://github.com/monicacg1111/TFG. El código se ha implementado en el lenguaje de programación Python, mediante el uso de *Jupyter Notebook*. En la figura B.1 podemos ver la estructura del proyecto:

```
TFG
├── Notebooks
│   ├── Visualizacion.ipynb
│   ├── Clasificacion.ipynb
│   ├── Clasificacion_EnronSpam.ipynb
│   ├── Clasificacion_LingSpam.ipynb
│   └── Clasificacion_SpamAssassin.ipynb
├── datos
│   ├── completeSpamAssassin.csv
│   ├── enronSpamSubset.csv
│   └── lingSpam.csv
└── img
    ├── unigrama1.png
    ├── wordcloud1.png
    └── ...
```

Figura B.1.: Estructura de la experimentación

- En la carpeta `Notebooks` encontramos los distintos cuadernos:
  - `Visualizacion.ipynb`: aquí realizamos el análisis exploratorio y visualización de los datos, de donde extraemos las figuras y gráficos necesarios para extraer patrones de nuestros datos y su contenido. Además de la visualización, también se realiza la primera fase de proprocesamiento o limpieza de datos.
  - `Clasificacion.ipynb`: este es el cuaderno general donde realizamos la clasificación del conjunto de datos unificado. Aquí realizamos el resto del procesamiento y el entrenamiento, validación y evaluación de los modelso de clasificación.
  - `Clasificacion_EnronSpam.ipynb`: cuaderno particular donde se realizan los mismos procedimientos que en `Clasificacion.ipynb`, pero únicamente para el conjunto de datos extraído de `enronSpamSubset.csv`.
  - `Clasificacion_LingSpam.ipynb`: cuaderno particular donde se realizan los mismos procedimientos que en `Clasificacion.ipynb`, pero únicamente para el conjunto de datos extraído de `enronSpamSubset.csv`.

    - `Clasificacion_SpamAssassin.ipynb`: cuaderno particular donde se realizan los mismos procedimientos que en `Clasificacion.ipynb`, pero únicamente para el conjunto de datos extraído de `completeSpamAssassin.csv`.

- En el directorio `datos` encontramos los ficheros de datos que leemos en los cuadernos, en formato `.csv`.
- En el directorio `img` encontramos las imágenes generadas en los cuadernos y también las usadas en la memoria.

B.2. Bibliotecas usadas

B.2.1. Pandas

Pandas [57] es una biblioteca de Python utilizada principalmente para manipulación y análisis de datos. Ofrece estructuras de datos flexibles y herramientas para trabajar con datos tabulares y de series temporales. Estas funciones son fundamentales para realizar tareas comunes de limpieza y manipulación de datos. Algunas de las funciones que hemos empleado son:

- `read_csv`: Una función utilizada para leer datos desde archivos CSV y cargarlos en un DataFrame de Pandas. Esto permite la manipulación y análisis de datos de manera conveniente en Python.
- `drop_duplicates`: permite eliminar las filas duplicadas de un DataFrame. Esto puede ser útil cuando se trabaja con conjuntos de datos que pueden contener duplicados y se preservar la unicidad de las instancias para análisis precisos.
- `dropna`: sirve para eliminar filas o columnas que contienen valores nulos (NaN) en un DataFrame. Crucial para mantener la calidad del conjunto de datos, eliminando observaciones incompletas o no válidas.
- `replace`: Permite sustituir valores en un DataFrame, utilizada en nuestro código para reemplazar espacios en blanco con NaN, facilitando la limpieza de datos.
- `str.lower`, `str.replace`, y `str.contains`: Métodos de la clase `str` para manipular cadenas de texto en DataFrames, esenciales para normalizar y filtrar datos textuales.
- `describe`: genera las estadísticas descriptivas de un DataFrame. Estas incluyen un resumen de la tendencia de los datos, su dispersión y la forma de su distribución, excluyendo los valores NaN.

B.2.2. Scikit-learn

Scikit-learn [58] es una biblioteca de aprendizaje automático de Python que ofrece una amplia gama de herramientas para tareas como clasificación, regresión, clustering y más. A continuación, enumeramos algunas de las funciones que hemos empleado en este trabajo:

- `ENGLISH_STOP_WORDS`: Una lista predeterminada de palabras vacías en inglés, como artículos, pronombres y preposiciones, utilizada para la eliminación de palabras vacías en el procesamiento de texto.

- `CountVectorizer`: Una función utilizada para convertir una colección de documentos de texto en una matriz de recuentos de términos/palabras.
- `TfidfVectorizer`: Similar a `CountVectorizer`, pero va un paso más allá al ponderar los términos según la frecuencia inversa del documento (TF-IDF). Esto ayuda a resaltar palabras que son más únicas para un documento, proporcionando una mejor discriminación.
- `train_test_split`: Una función utilizada para dividir los datos en conjuntos de entrenamiento y prueba. Con el parámetro `test_size` se indica la proporción de datos en los conjuntos train y test.
- `StratifiedKFold`: Una variante de `K-Fold cross-validation` que se utiliza para asegurar que cada pliegue del dataset de entrenamiento y validación represente correctamente las proporciones de las diferentes clases del conjunto completo de datos. De esta forma, cada conjunto de entrenamiento y validación resultante mantiene aproximadamente la misma proporción de clases que el conjunto original, lo que ayuda a evitar sesgos en la distribución de las clases.
- `MultinomialNB`: Una implementación del clasificador Naive Bayes multinomial en scikit-learn, adecuado para clasificación de textos con características discretas como recuentos de palabras.

B.2.3. String

El módulo `string` en Python proporciona acceso a operaciones comunes con caracteres de texto, incluyendo constantes útiles y funciones:

- `string.punctuation`: cadana preinicializada que devuelve todos los signos de puntuación.

B.2.4. Regular Expressions (re)

La biblioteca `re` de Python permite la manipulación y el manejo de cadenas de texto mediante expresiones regulares, ofreciendo una herramienta poderosa para la búsqueda y modificación de patrones complejos en texto:

- `re.sub`: Función que toma un string y un patrón, y devuelve otro string con los términos reemplazados por otros valores según el patrón. Utilizada en nuestro código para sustituir URLs y caracteres no alfanuméricos en los textos, facilitando la estandarización y limpieza de los datos antes del análisis.

B.2.5. NLTK

NLTK (Natural Language Toolkit) [59] es una biblioteca líder para el procesamiento de lenguaje natural (NLP) en Python. Ofrece múltiples de herramientas de procesamiento de texto para la clasificación, tokenización, derivación, etiquetado, análisis sintáctico y razonamiento semántico. Algunas de las funciones que hemos empleado en este trabajo son:

- `word_tokenize`: Una función utilizada para dividir textos en palabras o tokens. Es fundamental en el preprocesamiento de texto para la mayoría de las tareas de NLP, ya que convierte cadenas de texto largas en piezas manejables y analizables.
- `sent_tokenize`: Función para dividir un texto en oraciones.
- `bigrams`: Función utilizada para generar pares de palabras consecutivas (bigramas) de un listado de tokens. Devuelve un iterador de tuplas de dos elementos. Esta herramienta es esencial para analizar la estructura lingüística y el contexto en el texto, permitiendo estudiar la ocurrencia y las dependencias locales entre palabras.
- `trigrams`: Función utilizada para generar secuencias de tres de palabras consecutivas (trigramas) de un listado de tokens. Devuelve un iterador de tuplas de tres elementos.
- `synsets()`: Método utilizado para obtener todos los conjuntos de sinónimos (*synsets*) asociados a una palabra, facilitando el acceso a su estructura semántica dentro del léxico de `WordNet`.
- `hypernyms()`: Método empleado para obtener los hiperónimos de un *synset*, que son términos más generales relacionados con el concepto inicial. Esta funcionalidad permite ascender en la jerarquía semántica de `WordNet`, para identificar categorías superiores.

B.2.6. Wordcloud

La biblioteca `wordcloud` de Python es una herramienta que permite la visualización de palabras clave en forma de nube de palabras, donde la frecuencia e importancia de cada palabra influye en su tamaño en la visualización. En nuestro proyecto, se utilizó la siguiente función:

- `WordCloud`: Esta función genera una imagen de nube de palabras a partir de un texto. La función permite especificar parámetros como el número máximo de palabras, escalas de colores, filtros de frecuencia mínima y máxima, y dimensiones de la imagen. El tamaño de cada palabra en la nube es proporcional a su frecuencia relativa en el texto proporcionado.

B.2.7. Collections

El módulo `collections` en Python es parte de la biblioteca estándar y ofrece alternativas especializadas a los contenedores de datos built-in como diccionarios, listas, y tuplas. Dentro de este módulo, diversas estructuras de datos están diseñadas para proporcionar funcionalidades adicionales y optimizaciones. Una de las clases más utilizadas en este módulo es:

- `Counter`: es una subclase de diccionario que se utiliza para contar objetos hashables. Es especialmente útil en el contexto de procesamiento de lenguaje natural para contar la frecuencia de aparición de palabras o n-gramas en el texto.

B.2.8. URLExtract

La biblioteca `URLExtract` está diseñada para extraer URLs de textos basándose en patrones de identificación de URLs. Proporciona funcionalidades específicas para detectar y extraer enlaces web de un flujo de texto:

- `has_urls`: Esta función verifica si una cadena de texto contiene una o más URLs. Utiliza patrones y heurísticas definidas para identificar partes de texto que corresponden a enlaces web según estándares de formato de URLs.
- `find_urls`: Esta función extrae todas las URLs encontradas en una cadena de texto. Devuelve una lista de URLs identificadas, facilitando su análisis o manipulación posterior. Esta función es especialmente útil en contextos donde es necesario separar o analizar los enlaces contenidos en el texto de manera exhaustiva.

B.2.9. Imbalanced Learn (imblearn)

La biblioteca `imbalanced-learn` de Python ofrece soluciones para manejar conjuntos de datos con clases desbalanceadas. Está construida sobre `scikit-learn` y proporciona una variedad de métodos para reequilibrar las clases mediante técnicas de sobremuestreo y submuestreo. Algunas de las funciones y herramientas que hemos empleado en este trabajo son:

- `RandomOverSampler`: Función utilizada para realizar un sobremuestreo de la clase minoritaria. Funciona replicando aleatoriamente las instancias de la clase minoritaria hasta alcanzar un balance deseado entre las clases. Esto ayuda a mitigar el problema de sesgo hacia la clase mayoritaria al proporcionar más ejemplos de la clase minoritaria al modelo.

Glosario

A continuación definimos algunos conceptos matemáticos no explicados en la memoria, pero que se han mencionado en el trabajo.

Definición B.1. El producto escalar entre dos vectores $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ se define como

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i,$$

donde a_i y b_i son las componentes de $\mathbf{a}$ y $\mathbf{b}$, respectivamente.

Definición B.2. Sea $X \subseteq \mathbb{R}^d$. Decimos que X es compacto si para toda familia $\{O_i\}_{i \in I}$ de abiertos de $\mathbb{R}^d$ tal que $X \subseteq \bigcup_{i \in I} O_i$, se verifica que existen $i_1, \ldots, i_n \in I$ tales que $X \subseteq O_{i_1} \cup \ldots \cup O_{i_n}$. En otras palabras, X compacto si de cada recubrimiento por abiertos de X se puede extraer un subrecubrimiento finito.

Definición B.3. Sea $X \subseteq \mathbb{R}^d$ compacto. $L_2(X)$ es el espacio vectorial de las funciones cuadradas integrables sobre dicho conjunto, es decir,

$$L_2(X) = \left\{ f : \int_X f(x)^2 \, dx < \infty \right\}.$$

Definición B.4. Sea $X \subseteq \mathbb{R}^d$ y $\{f_n\} : X \to \mathbb{R}^m$ una sucesión de funciones. Se dice que f_n converge uniformemente a la función f definida en X, si para todo $\epsilon > 0$ existe un número natural n_0 tal que si $n \geq n_0$, entonces $|f_n(x) - f(x)| < \epsilon$, $\forall x \in X$.

Definición B.5. Una sucesión $\{h_n\}_{n=1}^{\infty}$ es de Cauchy si satisface la siguiente propiedad:

$$\sup_{m > n} \|h_n - h_m\| \to 0, \text{ cuando } n \to \infty.$$

Definición B.6. Un espacio F es completo si toda sucesión de Cauchy $\{h_n\}_{n=1}^{\infty}$ de elementos de F converge a un elemento $h \in F$.

Definición B.7. Un espacio F es separable si para cualquier $\epsilon > 0$ existe un conjunto finito de elementos $\{h_1, \ldots, h_n\}$ de F tal que para todo $h \in F$,

$$\min_{1 \leq i \leq n} \|h - h_i\| < \epsilon.$$

Definición B.8. Un espacio de Hilbert F es un espacio vectorial provisto de un producto escalar, que cumple ser completo y separable.

Bibliografía

[1] Usama Fayyad, Gregory Piatetsky-Shapiro, y Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, nov 1996.

[2] Uffe Kjærulff y Anders L. Madsen. Probabilistic networks - an introduction to bayesian networks and influence diagrams. Aalborg, Denmark, 2005. Aalborg University.

[3] Mohammed J. Zaki y Wagner Meira. *Data mining and analysis: Fundamental concepts and algorithms*. Cambridge University Press, Cambridge, England, 2014.

[4] Kang Zhao, Ling Xing, y Honghai Wu. S3uca: Soft-margin support vector machine-based social network user credibility assessment method. *Mobile Information Systems*, 2021:1–10, 2021.

[5] Romain Hardy. Understanding the kernel trick. https://romainlhardy.medium.com/understanding-the-kernel-trick-a787821d8580, 2020. Accessed: 2024-6-25.

[6] Arpit Jain. Support vector machines. https://medium.com/@arpitjainn2205/support-vector-machines-ab89fae48792, December 2022. Accessed: 2024-6-30.

[7] Holdout data and cross-validation. https://help.qlik.com/en-US/cloud-services/Subsystems/Hub/Content/Sense_Hub/AutoML/holdout-crossvalidation.htm. Accessed: 2024-5-20.

[8] Doug Steen. Understanding the ROC curve and AUC. https://towardsdatascience.com/understanding-the-roc-curve-and-auc-dd4f9a192ecb, September 2020. Accessed: 2024-7-6.

[9] María del Consuelo Justicia de la Torre. *Nuevas técnicas de minería de textos: Aplicaciones*. PhD thesis, Universidad de Granada, 2017.

[10] María Novo-Lourés, Reyes Pavón, Rosalía Laza, David Ruano-Ordas, y Jose R Méndez. Using natural language preprocessing architecture (NLPA) for big data text sources. *Sci. Program.*, 2020:1–13, 2020.

[11] Natural Language Processing. https://www.deeplearning.ai/resources/natural-language-processing/, January 2023. Accessed: 2024-7-2.

[12] María Chantal Perez Hernández. Explotación de los córpora textuales informatizados para la creación de bases de datos terminológicas basadas en el conocimiento. *Estudios de lingüística del español*, 18, 2002.

[13] Thomas D. Wason. Dr. tom's taxonomy guide: Description, use and selections. *IMS Global Learning Consortium, Inc.*, 2006.

[14] Antonio Moreno Ortiz. Diseño e implementación de un lexicón computacional para lexicografía y traducción automática. *Estudios de lingüística del español*, 9, 2000.

[15] Juan Luis Castro. Apuntes de la asignatura Ingeniería del Conocimiento, grado en Ingeniería Informática, Universidad de Granada, Curso 22-23.

[16] Wikipedia contributors. Correo basura. https://es.wikipedia.org/w/index.php?title=Correo_basura&oldid=157238043. Accessed: 2024-2-26.

[17] Tatyana Kulikova. El spam y el phishing en 2022. https://securelist.lat/spam-phishing-scam-report-2022/97582/, February 2023. Accessed: 2024-3-4.

[18] Ivan Belcic. Qué es el spam: guía esencial para detectar y prevenir el spam. https://www.avast.com/es-es/c-spam, January 2020. Accessed: 2024-3-4.

[19] Phishing. https://www.incibe.es/aprendeciberseguridad/phishing. Accessed: 2024-3-22.

[20] José R Méndez, Florentino Fdez Riverola, Fernando Díaz, y Juan M Corchado. Sistemas inteligentes para la detección y filtrado de correo spam: una revisión. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 11(34):63–81, 2007.

[21] Emmanuel Gbenga Dada, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi, y Opeyemi Emmanuel Ajibuwa. Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 5(6):e01802, 2019.

[22] Wikipedia contributors. Nilsimsa hash. https://en.wikipedia.org/w/index.php?title=Nilsimsa_Hash&oldid=1211187431. Accessed: 2024-3-16.

[23] Apache SpamAssassin: Welcome. https://spamassassin.apache.org/. Accessed: 2024-2-26.

[24] Ion Androutsopoulos. *Learning to filter unsolicited commercial E-mail*. "DEMOKRITOS", National Center for Scientific Research.

[25] Gordon Parker Rios y Hongyuan Zha. Exploring support vector machines and random forests for spam detection. En *CEAS 2004 - First Conference on Email and Anti-Spam*, páginas 30–31, Mountain View, California, USA, 2004.

[26] Francisco Herrera Triguero. Apuntes de la asignatura Aprendizaje Automático, grado en Ingeniería Informática, Universidad de Granada, Curso 23-24.

[27] Emmanuel Anguiano-Hernández. Naive bayes multinomial para clasificación de texto usando un esquema de pesado por clases. páginas 1–8, 2009.

[28] Juan Antonio Maldonado Jurado. Apuntes de la asignatura Estadística Descriptiva e Introducción a la Probabilidad, doble grado en Ingeniería Informática y Matemáticas, Universidad de Granada, Curso 19-20.

[29] Francisco Javier García Castellano. *Modelos bayesianos para la clasificación supervisada. Aplicaciones al análisis de datos de expresión genética*. PhD thesis, Universidad de Granada, 2009.

[30] Antonio Jesús Rodríguez Salas. Apuntes de la asignatura Lógica y Métodos Discretos, grado en Ingeniería Informática, Universidad de Granada, Curso 20-21.

[31] Richard J Trudeau. *Introduction to graph theory*. Dover Publications, 2013.

[32] Eva Millán Valldeperas. *Sistema bayesiano para modelado del alumno*. PhD thesis, Universidad de Málaga, 2000.

[33] Víctor Robles Forcada. *Clasificación supervisada basada en redes bayesianas. Aplicación en biología computacional*. PhD thesis, Universidad Politécnica de Madrid, 2003.

[34] Thomas Mitchell. *Machine Learning*. McGraw-Hill Professional, New York, NY, 1997.

[35] Juan Manuel Cabrera Jiménez y Fabricio O. Pérez Pérez. Clasificación de documentos usando naive bayes multinomial y representaciones distribucionales. páginas 1–13, 2011.

[36] 1.9. Naive Bayes. https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed: 2024-5-14.

[37] Sklearn.Naive_bayes.MultinomialNB. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html. Accessed: 2024-2-9.

[38] Harry Zhang. The optimality of naive bayes. *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004*, 2, 2004.

[39] Stephen Boyd y Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, England, 2004.

[40] R Fletcher. *Practical methods of optimization*. John Wiley & Sons, Chichester, England, 2 edition, 2013.

[41] John Shawe-Taylor y Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, Cambridge, England, 2011.

[42] Wenjian Wang, Zongben Xu, Weizhen Lu, y Xiaoyun Zhang. Determination of the spread parameter in the gaussian kernel for classification and regression. *Neurocomputing*, 55(3):643–663, 2003. Evolving Solution with Neural Networks.

[43] Hsuan-Tien Lin y Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. *Neural Computation*, 06 2003.

[44] Nitisha Bharathi. Email Spam Dataset. https://www.kaggle.com/datasets/nitishabharathi/email-spam-dataset/version/1, 2020. Kaggle Dataset. Accessed: 2024-1-30.

[45] Ion Androutsopoulos, John Koutsias, Konstantinos Chandrinos, Georgios Paliouras, y Costantine Spyropoulos. An evaluation of naive bayesian anti-spam filtering. *CoRR*, cs.CL/0006013, 2000.

[46] Leslie Kaelbling y Melinda Gervasio. Enron email dataset. https://www.cs.cmu.edu/~./enron/. Accessed: 2024-1-30.

[47] Spam assassin dataset. https://spamassassin.apache.org/old/publiccorpus/. Accessed: 2024-1-30.

[48] sklearn.svm.SVC. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2024-7-5.

[49] Matrices dispersas. https://es.mathworks.com/help/matlab/sparse-matrices.html. Accessed: 2024-7-5.

[50] José Soto Mejía, Guillermo Roberto Solarte Martínez, y Luis Eduardo Muñoz Guerrero. Matrices dispersas descripcion y aplicaciones. *Scientia Et Technica*, 2013.

[51] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. En *AAAI-98 workshop on learning for text categorization*, volume 752, páginas 41–48. Madison, WI, 1998.

[52] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. En *European conference on machine learning*, páginas 137–142. Springer, 1998.

[53] Leen Al Qadi, Hozayfa El Rifai, Safa Obaid, y Ashraf Elnagar. A scalable shallow learning approach for tagging arabic news articles. *Jordanian Journal of Computers and Information Technology*, 06:1, 09 2020.

[54] Jorge Casillas. Apuntes de la asignatura Inteligencia de Negocio, grado en Ingeniería Informática, Universidad de Granada, Curso 23-24.

[55] María Novo Lourés. *Preprocesamiento inteligente para mejorar la precisión del filtrado de spam*. PhD thesis, Universidad de Vigo, 2021.

[56] NLTK :: Sample usage for wordnet. https://www.nltk.org/howto/wordnet.html. Accessed: 2024-7-9.

[57] DataFrame - pandas 2.2.0 documentation. https://pandas.pydata.org/docs/reference/frame.html. Accessed: 2024-2-11.

[58] Scikit-learn. https://scikit-learn.org/stable/index.html. Accessed: 2024-2-9.

[59] NLTK :: Natural language toolkit. https://www.nltk.org/index.html. Accessed: 2024-5-31.