Universidad de La Laguna

Máster Universitario en Ciberseguridad e Inteligencia de Datos

Etapas de un proyecto de Aprendizaje Automático. Clasificación de Incidencias

Steps of a Machine Learning Project. Ticket Classification

David Valverde Gómez

La Laguna, Julio 2023

Máster Universitario en Ciberseguridad e Inteligencia de Datos

Dña. Isabel Sánchez Berriel, con N.I.F. 42.885.838-S profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutora, y

D. José Luis Roda García, con N.I.F. 43.356.123-L profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como cotutor

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

Que la presente memoria titulada:
“Etapas de un proyecto de Aprendizaje Automático. Clasificación de incidencias”

ha sido realizada bajo su dirección por **D.David Valverde Gómez**, con N.I.F. 79.081.228-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de Julio de 2023.

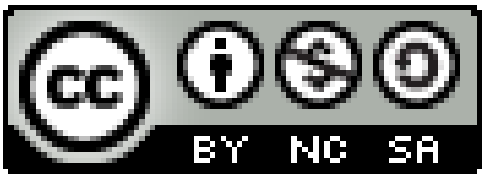
Agradecimientos

A Isabel y José Luis, por su apoyo en este proyecto,
pero sobre todo por su paciencia y buena disposición.

A mis amigos, compañeros de risas y de penas durante estos años.

Y a mi familia, siempre mi apoyo incondicional en todo lo que hago.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.

Resumen

El Aprendizaje Automático (Machine Learning, ML) es una de las ramas de la Inteligencia Artificial enfocada en el desarrollo de modelos capaces de aprender a partir de datos. Este enfoque revolucionario ha impulsado avances significativos en áreas como la visión por computador, la recomendación de contenido o el procesamiento del lenguaje natural, entre otros.

El objetivo de este trabajo es el estudio detallado del proceso a seguir en un proyecto real de Machine Learning, en compañía de un ejemplo práctico que trata de la construcción de una aplicación para la clasificación de incidencias del día a día de una empresa.

Para este trabajo se ha utilizado el lenguaje de programación Python, junto con diversas librerías que facilitan el tratamiento de los datos y la aplicación de diferentes técnicas y algoritmos, en un entorno de Jupyter.

Palabras clave: Aprendizaje Automático, Procesamiento del Lenguaje Natural, Python, Jupyter, clasificación de incidencias, preprocesado de datos, vectorización

Abstract

Machine Learning (ML) is a branch of Artificial Intelligence (AI) focused on developing models capable of learning from data. This revolutionary approach has driven significant advances in areas such as computer vision, content recommendation, and natural language processing, among others.

The goal of this work is to provide a detailed study of the process involved in a real-world Machine Learning project, accompanied by a practical example that focuses on building an application for the classification of everyday incidents in a company.

For this project, the Python programming language has been used along with various libraries that facilitate data processing and the implementation of different techniques and algorithms, within a Jupyter environment.

Palabras clave: Machine Learning, Natural Language Processing, Python, Jupyter, ticket classification, data preprocessing, vectorization

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

1.1. Motivación

El Aprendizaje Automático (Machine Learning, ML) [1] es un campo científico, subcategoría de la Inteligencia Artificial y relacionado con el Big Data [2], cuyo objetivo, como su propio nombre indica, es que una máquina sea capaz de aprender, o lo que es lo mismo, detectar patrones en la información.

De esta manera, un modelo informático con una gran cantidad de datos podría entrenarse hasta encontrar un patrón, y así poder incluso predecir datos futuros o realizar clasificaciones con una gran fiabilidad.

El Procesamiento del Lenguaje Natural (PLN) es una de las ramas de este campo, que se centra en el estudio del lenguaje, teniendo gran cantidad de aplicaciones como la generación automática de textos (la más conocida actualmente debido al enorme éxito que ha supuesto la publicación de ChatGPT [3]) o la clasificación de textos y documentos.

La aplicación del PLN para la clasificación de documentos es, sin lugar a dudas, una herramienta poderosa y con una enorme capacidad, que ya se aprecia a día de hoy, por ejemplo, en sistemas recomendadores [4] utilizados en populares aplicaciones de streaming de vídeo y música, como Netflix o Youtube para mejorar la experiencia de usuario y, por tanto, su satisfacción.

En el ámbito empresarial también presenta un gran potencial, como el análisis de sentimientos, el análisis de documentos legales o la clasificación de incidencias.

Esta última aplicación del PLN en empresas es la que motiva este trabajo, ya que la correcta clasificación de las incidencias que recibe una empresa, tanto de manera externa como interna, es un paso crucial que puede costar a los usuarios mucho tiempo y esfuerzo de realizarse de forma manual, mientras que un buen proyecto de PLN podría facilitar, e incluso mejorar este proceso de manera exponencial.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Máster es el estudio general de un proyecto de ML, concretamente, de un proyecto de Procesamiento de Lenguaje Natural. Durante este estudio, se analizará el ciclo de vida estándar que todo proyecto debe tener, se analizarán sus diferentes pasos y las técnicas y algoritmos que deben emplearse en cada uno de ellos.

Además, se realizará, como ejemplo práctico, un proyecto de clasificación de incidencias en colaboración con la empresa iTop S.L., que ha proporcionado un conjunto de datos de prueba, donde se detalla cada uno de los pasos a seguir y se llevará a cabo una comparativa de posibles técnicas y algoritmos a utilizar en cada etapa del proyecto.

1.3. Estructura del documento

Este documento está estructurado de la siguiente forma:

- **Capítulo 1. Introducción.**

  Se indican las motivaciones y objetivos del proyecto.

- **Capítulo 2. Antecedentes y estado del arte.**

  Estudio de la historia y situación actual del Machine Learning, el Procesamiento del Lenguaje Natural y algunas consideraciones a tener en cuenta en un proyecto real de ML.

- **Capítulo 3. Tecnologías utilizadas.**

  Se indican los recursos empleados en el proyecto, tanto software como hardware.

- **Capítulo 4. Desarrollo.**

  Se explica cada uno de los pasos realizados para la finalización del proyecto.

- **Capítulo 5. Resultados.**

  Se muestran y analizan los resultados del trabajo realizado.

- **Capítulo 6. Presupuesto.**

  Estimación del coste que supondría el desarrollo de este proyecto.

- **Capítulo 7. Conclusiones y líneas futuras.**

- **Capítulo 8. Summary and Conclusions.**

Capítulo 2. Antecedentes y Estado del Arte

2.1. Aprendizaje Automático

El Machine Learning, o aprendizaje automático, es una rama de la Inteligencia Artificial (IA) que se ocupa del desarrollo de algoritmos y técnicas que permiten a los programas informáticos aprender a través de su propia experiencia, y ser capaces de mejorar su rendimiento en ciertas tareas sin necesidad de ser reprogramados.

Este aprendizaje se lleva a cabo a través de la extracción de información valiosa y la detección de patrones sobre los datos que se reciben, ya sean números, textos o imágenes.

Esta disciplina ha experimentado un enorme desarrollo en los últimos años, debido en gran medida a dos causas principales.

Por una parte, tanto los avances en hardware, con la mejora de las GPUs [5] (Unidades de Procesamiento Gráfico) y la creación de las TPUs [6] (Unidades de Procesamiento Tensorial), como la programación de bibliotecas de código abierto como TensorFlow [7] han ayudado enormemente a facilitar, acelerar y mejorar la creación y entrenamiento de modelos de aprendizaje.

Por otro lado, el aumento de la disponibilidad de grandes volúmenes de datos y la mejora de técnicas de recolección y almacenamiento gracias al Big Data, permite a los modelos alimentarse de datos más abundantes y diversos, aprendiendo así patrones más complejos y adaptándose mejor a situaciones nuevas.

Además, el ML presenta una gran cantidad de aplicaciones, entre las que destacan la generación automática de textos (la que ha ganado notoriedad con la publicación de ChatGPT), la clasificación de documentos, el reconocimiento de voz, la detección de spam o la personalización de recomendaciones en plataformas de streaming y comercio electrónico. En resumen, el Machine Learning tiene un amplio alcance y sigue encontrando nuevas formas de mejorar y automatizar diversas tareas en múltiples industrias.

Es esta enorme potencialidad la que hace que, a día de hoy, sea imprescindible para cualquier informático estar familiarizado con este concepto y con los pasos a seguir para desarrollar un adecuado proyecto de aprendizaje automático.

2.2. Procesamiento del Lenguaje Natural

El PLN es una rama del aprendizaje automático cuyo cometido es otorgar a los programas informáticos la capacidad de comprender textos y palabras de la misma forma que los humanos.

Para ello, se combina lingüística computacional con modelos de ML, para procesar el lenguaje en forma de datos que la máquina entienda, y extraer la información necesaria para “comprender” el significado, la intención y el sentimiento del emisor [8].

El avance del PLN en los últimos años ha ido de la mano con el ya comentado desarrollo del ML, logrando una drástica mejora en la capacidad de comprensión del lenguaje natural por parte de las máquinas. Los modelos de PLN han demostrado una mayor precisión en tareas como la clasificación de documentos, la extracción de información o el análisis de sentimientos.

Otro avance importante en el desarrollo del PLN ha sido el surgimiento de modelos de generación de texto. Estos modelos son capaces de producir texto coherente y relevante, lo que ha abierto nuevas posibilidades en áreas como la redacción automática, la asistencia en la redacción de contenido y la creación de chatbots más sofisticados y conversacionales.

Todas estas mejoras en las capacidades de este campo ya se ven en ámbitos reales tales como la atención al cliente, con la implementación de chatbots para respuestas automatizadas, o en la medicina, para ayudar en el diagnóstico de enfermedades.

Ya se ha demostrado que la utilización de algoritmos de PLN resulta de vital importancia para la mejora de la productividad de las empresas a día de hoy, no solo minimizando el trabajo manual sino también mejorando sus resultados.

2.3. Consideraciones para un proyecto de ML

La mayoría de los proyectos de Machine Learning que se llevan a cabo en empresas terminan sin éxito o con resultados infructuosos. Esto se debe a que muchos de estos proyectos se inician sin un criterio claro de éxito, esperan cumplir objetivos que realmente son impracticables, o no se gestionan correctamente [9].

Por esto, para mejorar las posibilidades de éxito de un proyecto de ML, hay ciertas consideraciones que deben tenerse en cuenta.

2.3.1. Priorizar un proyecto sobre otro

Cuando se tienen múltiples posibles proyectos a llevar a cabo, es importante tener en cuenta una serie de aspectos a la hora de elegir cuál desarrollar.

Valorar el impacto que puede tener el proyecto en la organización, posibles beneficios económicos, mejora en la eficiencia, satisfacción del cliente… Se debe priorizar aquellos trabajos en los que incluso las soluciones que no son perfectas provocan una importante mejora en la empresa.

La viabilidad del producto también debe evaluarse. Siempre se preferirán los proyectos en los que su desarrollo y la obtención de resultados satisfactorios son factibles y requieren un tiempo y esfuerzo menor. Los programas con expectativas irreales están condenados al fracaso.

Deben evitarse productos con condiciones que generalmente suponen problemas en el ML, como salidas complejas, el requerimiento de confianza absoluta o la expectativa de generalización de los resultados.

2.3.2. Arquetipos

Un arquetipo [10] es un modelo a partir del cual pueden desarrollarse sistemas o aplicaciones informáticas. Sirven como plantilla y ayudan a los programadores a seguir una serie de pautas predefinidas.

Los desarrollos de Machine Learning pueden clasificarse en distintos arquetipos, en función de la finalidad de los mismos. Tres arquetipos comunes son:

- Proyectos que mejoran un proceso existente. Por ejemplo, optimizar rutas de transporte o mejorar las recomendaciones de una plataforma de streaming.
- Proyectos que complementan un proceso manual. Como convertir diseños gráficos en interfaces de aplicaciones, construir un programa de autocompletado de texto o el análisis de sentimientos en redes sociales.
- Proyectos que automatizan un proceso manual. Esto puede incluir el desarrollo de vehículos autónomos, la automatización del servicio a clientes o la clasificación automática de documentos.

En todos estos arquetipos es importante tener en cuenta el concepto de “Data Flywheels”, utilizado para describir un ciclo virtuoso de generación y aprovechamiento de datos. Hace referencia al proceso en que la información que se genera en una etapa del proyecto es utilizada para mejorar de forma iterativa los pasos siguientes, creando así un ciclo continuo de mejora y retroalimentación.

En este contexto, se considera que en cada ciclo la calidad y valor de los datos se incrementa, lo que conlleva también una mejora en la precisión y efectividad de los productos basados en datos. Esto puede aplicarse en diversos campos, como el desarrollo de productos, la optimización de algoritmos de aprendizaje automático o la toma de decisiones basada en datos, entre otros.

2.3.3. Métricas

Las métricas son herramientas fundamentales para los proyectos de aprendizaje automático, ya que se emplean para evaluar el rendimiento y medir el éxito que se obtiene con un modelo. En la mayoría de los casos reales, se tienen múltiples métricas para analizar, y dado que un sistema de ML trabaja mejor a la hora de optimizar un único valor, es recomendable encontrar la manera de combinar las métricas de interés en una sola, a través de alguna fórmula concreta o un promedio entre ellas.

Existen diferentes enfoques para abordar esta combinación de métricas. El primero y más sencillo es calcular un promedio, simple o ponderado, de todas las métricas.

El segundo enfoque implica la selección de una métrica concreta como umbral, y evaluar el modelo en base a ese valor de umbral. La elección de esta métrica crítica depende del ámbito concreto del trabajo, pero por lo general se opta por aquellas que son menos sensibles a la elección del modelo a aplicar.

La ventaja de este enfoque radica en su capacidad para establecer un punto de referencia claro para evaluar el rendimiento del modelo. Al fijar un umbral específico, se puede determinar si el modelo está cumpliendo con los criterios establecidos y si alcanza el nivel de desempeño deseado.

Otro abordaje consiste en utilizar una fórmula más compleja o específica del dominio del problema. Para esto se debe seguir un proceso sólido que evalúe todos los requisitos del proyecto y el rendimiento actual del modelo con respecto a estos requisitos, para así localizar las métricas específicas que necesitan mejoras o ajustes para un rendimiento óptimo. Este proceso de ajuste de medidas se repite hasta que los resultados obtenidos cumplan los requisitos del problema.

La elección de una métrica adecuada resulta de vital importancia en este tipo de emprendimientos para determinar un objetivo claro y detectar de forma precisa si se cumplen los requisitos iniciales del problema. Las más comunes son la precisión, la exhaustividad o *recall*, la exactitud o *accuracy* y la puntuación F1 o *F1 score,* entre otras.

2.3.4. Líneas Base

Las líneas base o baselines de un proyecto de aprendizaje automático son modelos simples que se utilizan como punto de referencia para evaluar las métricas de modelos más complejos. Sirve como límite inferior para el rendimiento esperado de un modelo.

La elección de una base adecuada depende del ámbito del problema a resolver. Pueden ser externas, como requisitos empresariales, o internas, como el rendimiento humano de los trabajadores.

Por ejemplo, se busca construir un modelo que automatice la clasificación de imágenes médicas para detectar ciertas enfermedades. Si un médico humano tiene una precisión de un 80% en esta tarea, no sería aceptable un modelo que no alcance ese porcentaje, y habría que rediseñarlo o adaptar mejor sus parámetros.

Disponer de un conjunto de líneas base para los diferentes tipos de proyectos sería de gran valor al no tener que partir de cero ni tener que “reinventar la rueda” a la hora de comenzar nuevos desarrollos.

2.3.5. Ciclo de Vida

Todo buen proyecto de ML, independientemente de su ámbito o características, debe seguir un esquema estándar que define de forma general los pasos a realizar para su correcta puesta en producción [11].

Planteamiento del problema. En primer lugar, se debe realizar un planteamiento preciso del problema a resolver. Esto implica entender los datos, establecer unos objetivos concretos, definir las métricas que se van a utilizar para evaluar el desempeño del modelo y realizar una evaluación de la solución que existe actualmente para concretar una línea base de referencia, es decir, el mínimo desempeño que se espera del producto.

Recolección y etiquetado de los datos. Se debe intentar recopilar la mayor cantidad de datos posible y que abarque todas las posibles casuísticas del problema. Es importante construir un dataset amplio y bien estructurado para poder extraer toda la información necesaria.

Análisis exploratorio y preprocesado de datos. Con el fin de comprender al máximo su estructura y arreglar errores que puedan haberse ocasionado durante su construcción: tratamiento de valores faltantes o nulos, formato adecuado de los datos, rebalanceo de clases o tratamiento de textos para disminuir su variabilidad (lematización [12]) y eliminar palabras que no aporten significado (stop words [13]).

Extracción de características. Donde se extrae de los datos la información más relevante o con mayor impacto para el modelo. En este paso se aplicarían técnicas como el análisis de componentes principales o la creación de nuevas características como combinación de las existentes. En proyectos concretos de procesamiento del lenguaje natural, se suele utilizar la vectorización, que es un proceso por el cuál se representan textos como vectores numéricos, convirtiendo así su información a un formato que un algoritmo pueda entender.

Construcción del modelo. Consiste en el entrenamiento y ajuste del algoritmo o los algoritmos elegidos. Se divide el conjunto de datos en un subconjunto de entrenamiento y otro de validación. Se utiliza el conjunto de entrenamiento para construir los modelos que posteriormente serán evaluados.

Evaluación del modelo. Se realiza sobre el conjunto de datos de validación, se valoran las métricas que se han definido al comienzo del proyecto y se verifica que se cumple con la línea base establecida. De ser así, se escoge el mejor algoritmo, en caso contrario, se vuelve al paso anterior para ajustar los parámetros de los modelos para optimizar los mismos y mejorar los resultados.

Puesta en producción. Tras la validación y elección del mejor algoritmo, se pone el proyecto en producción, es decir, se integra el modelo construido en el sistema de destino y se despliega para su uso.

Mantenimiento. Una vez puesto en producción, se inicia una etapa de monitorización y mantenimiento para detectar posibles errores o bajadas de rendimiento del producto, y actualizarlo o ajustarlo en caso de que sea necesario.

En la figura 2.1 se presentan gráficamente los pasos ya comentados para el desarrollo de un proyecto ML.

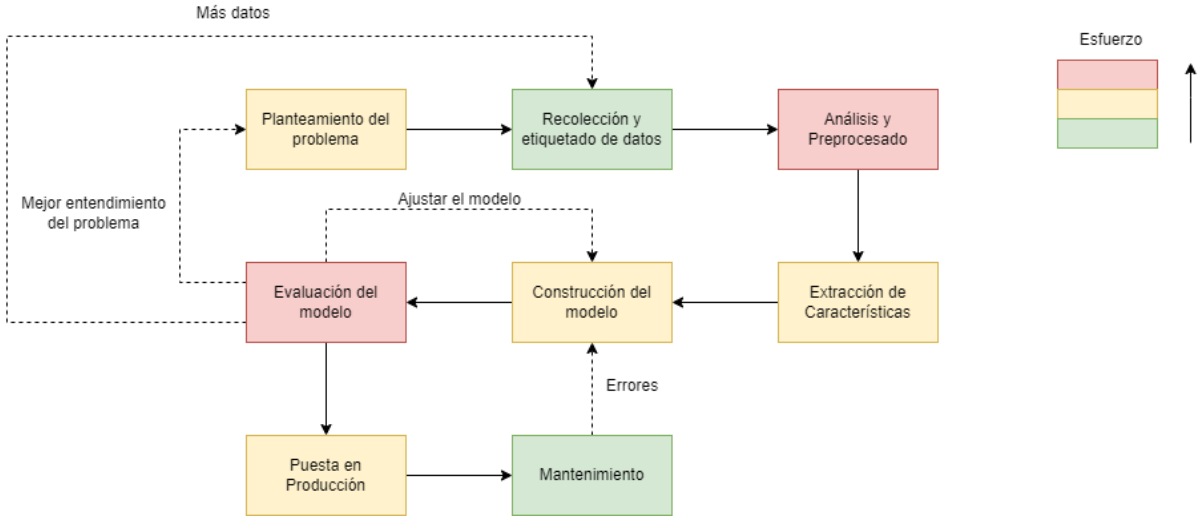


Figura 2.1.: Ciclo de vida de un proyecto ML

Capítulo 3. Tecnologías utilizadas

3.1. Hardware

No se ha requerido de ningún hardware concreto para desarrollar este trabajo, ya que el modelo generado ha sido construido utilizando procesamiento en la nube. Por lo que solo ha sido necesario mi ordenador portátil personal:

- Portátil Lenovo ideapad 530S-151KB con procesador Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99 GHz y 8GB de RAM.

3.2. Software

En cuanto a las aplicaciones software utilizadas, estas han sido diversas en las diferentes etapas del trabajo:

- Para el desarrollo del proyecto se ha utilizado:
  - Como entorno de desarrollo se han utilizado cuadernos de Jupyter en Google Colaboratory. [14]
  - El lenguaje de programación Python [15]. Versión 3.10.
  - Diversas librerías del sistema como *math* o *string.*
  - Librerías de manejo de datos como *pandas* o *numpy.*
  - Las librerías *matplotlib* y *seaborn* para la visualización de datos.
  - Librerías especializadas de machine learning como *scikit-learn* y *nltk.*
  - Múltiples librerías para la limpieza de los textos (*django, spacy*) y la vectorización (*gensim*).

- Para el almacenamiento de ficheros y documentos:
  - Google Drive [16]

- Para la redacción de la memoria:
  - Google Docs [17]

Capítulo 4. Desarrollo

4.1. Planteamiento del proyecto

La clasificación de incidencias (o tickets) en las empresas es un desafío común que enfrentan las organizaciones a la hora de gestionar el soporte técnico, el servicio al cliente o cualquier otro tipo de consultas relativas a su entorno empresarial.

Esta tarea consiste en categorizar y asignar etiquetas a los problemas reportados por clientes, empleados u otros usuarios, para que estos sean correctamente organizados y distribuidos a la sección o departamento de la empresa responsable de su resolución.

Habitualmente son empleados humanos quienes se encargan de asignar estas incidencias a un departamento u otro. Sin embargo, esta clasificación no es perfecta, y un ticket mal clasificado puede suponer enormes pérdidas de tiempo, ya que debe detectarse el error y volver a remitir la información a la sección correcta (suponiendo que no vuelva a clasificarse erróneamente). Así, este problema no sólo cuesta tiempo, sino también recursos y una decadencia en la satisfacción del cliente al aumentar la demora en la resolución de su problema.

Por esto, se pretende desarrollar un sistema clasificador de incidencias, que adapte los datos recogidos y construya un modelo de ML que sea capaz de clasificar una incidencia en función de su categoría o el producto al que pertenece, para poder ser correctamente distribuida al responsable correspondiente.

Haciendo referencia a lo estudiado en el apartado 2.3, se establece que:

- Se trata de un proyecto factible y viable, que puede conllevar un impacto elevado en la empresa.
- Se aplica al arquetipo de proyectos que automatizan un proceso manual.
- La métrica con la que se evaluará este proyecto será el *f1-score*, que combina la precisión y el *recall*, y es especialmente útil cuando se desea encontrar un equilibrio entre minimizar los falsos positivos y los falsos negativos. Se suele utilizar cuando el costo de los falsos positivos y falsos negativos es similarmente importante, como en este problema, ya que ambos casos radican en el mismo resultado.

- La eficiencia media de un empleado en este tipo de tareas puede variar significativamente en función de factores como su experiencia, o la complejidad de la incidencia, por lo que es difícil tomar un valor como referencia a la hora de establecer una línea base para la evaluación del modelo.

Este trabajo se ha desarrollado en colaboración con la empresa iTop S.L. [18], que se ha encargado de la recopilación y etiquetado de los datos y ha proporcionado un fichero con alrededor de 1000 incidencias correctamente clasificadas para poder entrenar el modelo.

Estos datos se han cargado en un cuaderno de Jupyter [19] sobre un entorno Python para dar comienzo al trabajo. Cabe destacar que, para evitar la divulgación de los datos utilizados, no se mostrará en ningún momento su contenido, a excepción de las columnas del dataset y las etiquetas a clasificar.

4.2. Análisis y Preprocesado de datos

La etapa de análisis exploratorio y preprocesado consiste en examinar los datos en bruto para identificar la información relevante que proporcionan, como sus campos, sus etiquetas o su formato, para posteriormente realizar aquellas acciones necesarias para adaptar la información a un formato adecuado para los modelos de machine learning a aplicar.

En primer lugar, se ha estudiado el contenido del dataset, así como sus columnas y su número de entradas.

```
[ ]   1 incidencias.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1049 entries, 0 to 1048
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Título            1049 non-null   object
 1   Descripción       947 non-null    object
 2   Solución          173 non-null    object
 3   Fecha de creación 1049 non-null   object
 4   Categoría         1042 non-null   object
 5   PRODUCTO          1049 non-null   object
dtypes: object(6)
memory usage: 49.3+ KB
```

Figura 4.1.: Información del Dataset

De esta información se observa un conjunto de datos con 1049 entradas y 6 columnas:

- Título de la incidencia
- Descripción de la incidencia
- Solución de la incidencia
- Fecha de creación
- Categoría asignada
- Producto relacionado

Tras el estudio de los datos (que no se mostrarán), destacan ciertos aspectos:

- La existencia de valores nulos
- Etiquetas HTML en los textos
- La fecha de creación es indiferente para la clasificación
- Aparentemente pueden clasificarse los tickets tanto por categoría (existen 4 categorías diferentes), como por producto (6 productos).

Las primeras modificaciones en el código consisten en la eliminación del campo ‘Fecha de creación’ que no aporta información relevante, y la limpieza de los textos, lo cual conlleva la eliminación de las etiquetas HTML presentes, sustituir los campos nulos (NaN) por textos vacíos (‘ ‘) y unificar los campos ‘Título’, ‘Descripción’ y ‘Solución’ en un único campo con todo el texto relativo a la incidencia.

El primer paso será eliminar los datos no necesarios, como el campo 'Fecha de creación'.

```
[5]  1 incidencias = incidencias.drop(['Fecha de creación'], axis=1)
```

Figura 4.2.: Eliminación del campo ‘Fecha de creación’

```
[6]  1 def limpiar_texto(texto):
     2   texto_limpio = strip_tags(texto)
     3
     4   # También se eliminan los saltos de línea
     5   texto_limpio = texto_limpio.replace('\n', '')
     6
     7   # En este punto, los valores NaN han sido convertidos a la cadena 'nan' automáticamente, por lo que también se eliminan
     8   texto_limpio = texto_limpio.replace('nan', '')
     9   return texto_limpio
    10
    11 incidencias['Descripción'] = incidencias['Descripción'].apply(limpiar_texto)
    12 incidencias['Solución'] = incidencias['Solución'].apply(limpiar_texto)
```

Una vez limpio el texto, se concatenan los 3 campos de texto en una nueva variable.

```
[7]  1 incidencias = incidencias.assign(texto_unificado=incidencias['Título'].astype(str) + ' ' +
     2                                  incidencias['Descripción'].astype(str) + ' ' + incidencias['Solución'].astype(str))
```

Figura 4.3.: Limpieza de texto

Tras la limpieza del texto, se detectó que el campo ‘Categoría’ presentaba 7 valores nulos. Al ser una cantidad muy pequeña, se decidió eliminar las entradas de datos correspondientes

Ahora se pasa al análisis de valores nulos, si existieran.

```
[8]  1 incidencias.isnull().sum()
```

```
Título            0
Descripción       0
Solución          0
Categoría         7
PRODUCTO          0
texto_unificado   0
dtype: int64
```

Se observan 7 valores nulos en la variable 'Categoría'. Como esta será una de las variables a predecir, y la cantidad de nulos es muy pequeña, simplemente se eliminarán estas entradas.

```
[9]  1 incidencias = incidencias.dropna()
     2 incidencias.isnull().sum()
```

```
Título            0
Descripción       0
Solución          0
Categoría         0
PRODUCTO          0
texto_unificado   0
dtype: int64
```

Figura 4.4.: Eliminación de valores nulos

Sigue el estudio del balance de clases, donde se analiza la distribución de los distintos valores que toman las dos variables a clasificar (categoría y producto).

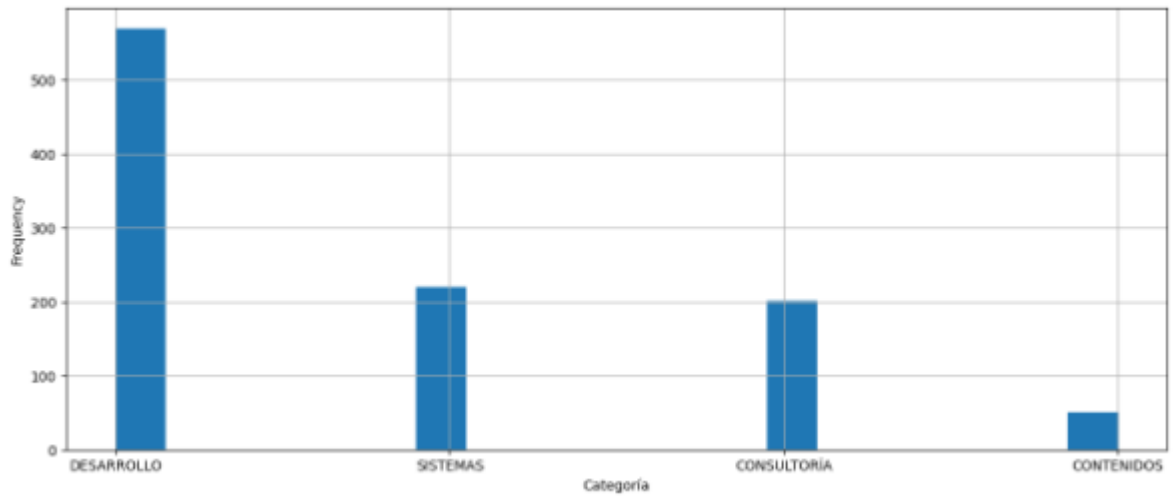


Figura 4.5.: Distribución de la variable ‘Categoría’

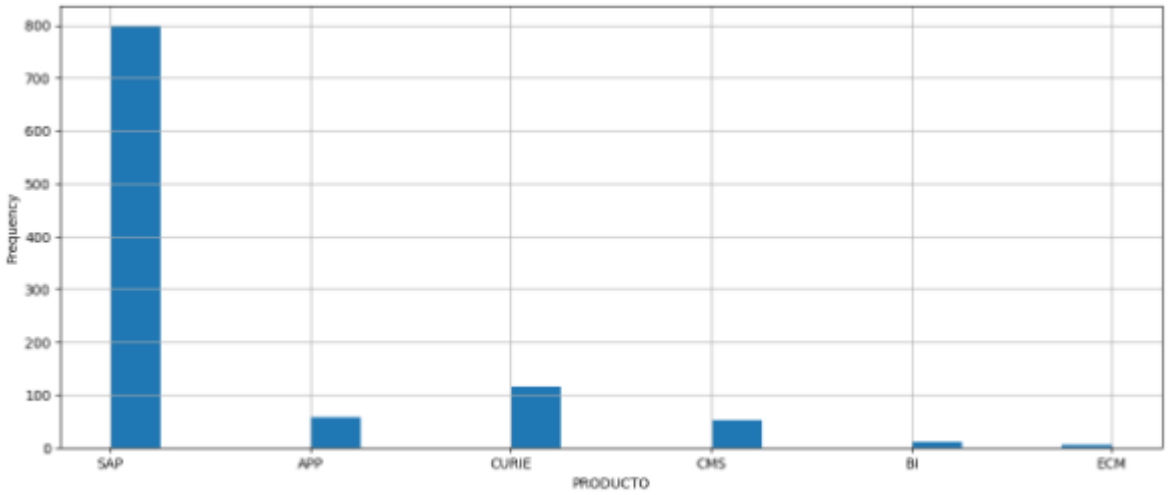


Figura 4.6.: Distribución de la variable ‘Producto’

Existe un enorme desbalanceo de clases en ambas variables, donde la categoría ‘Desarrollo’ y el producto ‘SAP’ suponen un 54.6% y un 76.48% de las incidencias respectivamente. Esto quiere decir que si, por ejemplo, se construyera un clasificador que siempre designara el producto ‘SAP’, tendría un 76% de precisión.

Esto supone un gran problema dado que los datos no permitirían al clasificador hacer una buena generalización de los casos, ya que este siempre tendería a clasificar las variables con una mayor frecuencia. Es por esto que deben aplicarse técnicas de rebalanceo para intentar igualar las distribuciones de ambos campos. No obstante, la técnica de rebalanceo a utilizar requiere que todos los campos sean numéricos, por lo que no podrá ejecutarse hasta después de haber realizado el paso de vectorización (representación numérica de los textos).

Dejando esto a un lado, se avanza al último paso de esta sección: la lematización de los textos relativos a cada incidencia. La lematización es un proceso de normalización de palabras cuyo objetivo principal es reducirlas a su base o forma canónica (lema), que representa el significado fundamental de la palabra.

La lematización es útil en el PLN porque reduce la variación léxica sin perder información y permite tratar las diferentes formas de una palabra como la misma entidad. Por ejemplo, las palabras "correr", "corro", "corres" y "corriendo" se lematizarían todas como "correr". Para este proceso se ha utilizado la librería *spacy* [20].

Se aprovecha este proceso para eliminar los signos de puntuación y las llamadas *stop words*: palabras que no aportan significado al contexto de la oración, tales como conectores, artículos, preposiciones o pronombres.

```python
def lemmatization_process(text):
    text = text.replace('.', ' ')
    # Tokenización
    tokens = word_tokenize(text)

    # Conversión a minúsculas
    tokens_lower = [token.lower() for token in tokens]

    # Eliminación de stopwords y puntuación
    tokens_filtered = [token for token in tokens_lower if token not in STOP_WORDS and token not in string.punctuation]
    text = " ".join(tokens_filtered)

    # Tokenización y lematización
    doc = nlp(text)
    lemmatized_words = [token.lemma_ for token in doc]

    # Filtrado de palabras resultantes
    final_words = [token.lower() for token in lemmatized_words if token.isalpha() and len(token) > 1]
    final_words = " ".join(final_words)
    return final_words

incidencias['texto_unificado_lematizado'] = incidencias['texto_unificado'].apply(lemmatization_process)
```

Figura 4.7.: Proceso de lematización

Una vez limpios y procesados los textos de las incidencias, se da por terminada esta etapa (a falta del proceso de rebalanceo de clases, que se realizará más adelante) y se da paso a la extracción de características de los datos.

4.3. Extracción de características

La extracción de características es un paso crucial en todo desarrollo de Machine Learning. Consiste en transformar los datos en un conjunto de características relevantes y representativas, de manera que la información sea fácilmente interpretable para el algoritmo a utilizar.

Pese a la existencia de múltiples técnicas que pueden aplicarse en esta etapa, este caso concreto se ha centrado en la vectorización del texto.

En el contexto del procesamiento del lenguaje natural (PLN), la vectorización de texto es esencial, ya que los algoritmos de aprendizaje automático generalmente requieren datos de entrada en forma numérica. A través de la vectorización, se transforman las palabras o documentos en vectores de números que capturan características y propiedades del texto.

Se aplicarán 3 métodos de vectorización diferentes, con el fin de realizar una comparativa entre ellos a través de los resultados de la clasificación:

- TF-IDF
- Word2Vec
- FastText

4.3.1. TF-IDF

TF-IDF [21] es una técnica de codificación de documentos, que asigna a cada palabra un valor en función de su importancia a la hora de distinguir entre ellos. El valor TF-IDF aumenta de manera proporcional al número de veces que una palabra aparece en un texto, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son, por lo general, más comunes que otras.

Esta técnica está formada por dos componentes, y su valor se extrae de la multiplicación entre ambos:

- TF (*Term Frequency* o Frecuencia de Término). Describe la frecuencia con que aparece una palabra en el mismo texto. Se calcula como la división entre el número de veces que se repite el término concreto en el documento, y el número total de palabras en el texto.

$$TF = \frac{Número\ de\ veces\ que\ el\ término\ aparece\ en\ el\ documento}{Número\ total\ de\ términos}$$

- IDF (*Inverse Document Frequency* o Frecuencia Inversa de Documento). Se emplea para disminuir la importancia de aquellas palabras que aparecen más a menudo en múltiples documentos. Se calcula como el logaritmo de la división entre el número total de documentos y el número de documentos en que aparece el término.

$$IDF = log(\frac{Número\ total\ de\ documentos}{Número\ de\ documentos\ que\ contienen\ el\ término})$$

El valor TF-IDF de cada palabra se calcula como la multiplicación de estos componentes:

$$TF - IDF = TF\ x\ IDF$$

Para la implementación de esta técnica en el código, se ha utilizado la función *TfidfVectorizer()* de la librería *sklearn* y se ha diseñado una función que construye un dataframe donde cada palabra se representa a través de una columna con su valor TF-IDF asociado para cada incidencia.

```python
def aplicar_tfidf(df):
    # Obtener la lista de descripciones
    descripciones = df['texto_unificado_lematizado'].tolist()

    # Crear una instancia de TfidfVectorizer
    vectorizer = TfidfVectorizer()

    # Aplicar TF-IDF a las descripciones
    matriz_tfidf = vectorizer.fit_transform(descripciones)

    # Obtener los términos (palabras) en el vocabulario
    vocabulario = vectorizer.get_feature_names_out()

    # Crear un diccionario con los valores TF-IDF de cada término para cada descripción
    datos_tfidf = matriz_tfidf.toarray()

    # Crear un nuevo DataFrame con las columnas del vocabulario y la categoría
    df_tfidf = pd.DataFrame(datos_tfidf, columns=vocabulario)

    df_tfidf['Categoría'] = df['Categoría']
    df_tfidf['PRODUCTO'] = df['PRODUCTO']

    return df_tfidf

# Aplicar TF-IDF al DataFrame
incidencias_tfidf = aplicar_tfidf(incidencias)

# Imprimir el DataFrame resultante
incidencias_tfidf
```

Figura 4.8.: Aplicación de TF-IDF

abran	...	índice	ópera	óptico	órdenes	única	únicamente	único	útil	Categoría	PRODUCTO
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	DESARROLLO	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	SISTEMAS	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	DESARROLLO	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	DESARROLLO	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	CONSULTORÍA	SAP
...	...	...	...	...	...	...	...	...	...	...	...
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	CONSULTORÍA	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	CONSULTORÍA	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	CONSULTORÍA	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	SISTEMAS	SAP
0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	CONSULTORÍA	SAP

Figura 4.9.: Resultado de TF-IDF

4.3.2. Word2Vec

Word2vec [22] es una técnica de inteligencia artificial que permite el análisis algorítmico de textos mediante la conversión de palabras en vectores numéricos.

El objetivo principal de Word2Vec es capturar las relaciones semánticas y similitudes entre palabras utilizando vectores numéricos densos como representación. Se entrena utilizando grandes cantidades de texto sin etiquetar y utiliza una arquitectura de deep learning llamada red neuronal de dos capas.

Durante el entrenamiento, el algoritmo ajusta los pesos de la red neuronal para que las representaciones vectoriales de palabras similares sean cercanas en el espacio vectorial. Esto significa que las palabras con significados o contextos similares tendrán vectores similares. Por ejemplo, las palabras "gato" y "felino" tendrán vectores cercanos.

Para la implementación de esta técnica se ha utilizado la función *Word2Vec()* de la librería *gensim* y se ha diseñado una función que convierta cada texto en un vector de palabras y aplique *Word2Vec()* a cada una. Sin embargo, esto resultará en que cada palabra se representará como un vector de tamaño 100, lo cuál no resulta factible a nivel computacional, por lo que se ha decidido convertir cada texto a un vector de tamaño 100 formado por la media de todos los vectores de las palabras que forman el documento.

```
# Este código transformará cada texto en un vector de palabras.
# Posteriormente, Word2Vec representará cada palabra como un vector de tamaño 100
# Esto no es manejable para los clasificadores, por lo que cada texto
# se representará como una media de todos los vectores de las palabras que lo forman (también de tamaño 100).
# Finalmente se desglosará el vector en un DF de 100 columnas + categoría + producto, para poder aplicar sobremuestreo

# Tokenización y creación de una lista de palabras
tokenized_texts = [text.lower().split() for text in incidencias['texto_unificado_lematizado']]

# Entrenamiento del modelo Word2Vec
model = Word2Vec(tokenized_texts, vector_size=100, min_count=1)

# Representación vectorial de los textos
text_vectors = []
for text in tokenized_texts:
    vectors = [model.wv[word] for word in text if word in model.wv]
    if vectors:
        text_vector = np.mean(vectors, axis=0)
        text_vectors.append(text_vector)
    else:
        # Si un texto no tiene ninguna palabra presente en el modelo Word2Vec, se asigna un vector de ceros
        text_vectors.append(np.zeros(100))

# Se construye el DF
incidencias_word2vec = pd.DataFrame(text_vectors)
incidencias_word2vec['Categoría'] = incidencias['Categoría']
incidencias_word2vec['PRODUCTO'] = incidencias['PRODUCTO']

incidencias_word2vec
```

Figura 4.10.: Implementación de Word2Vec

9	...	92	93	94	95	96	97	98	99	Categoría	PRODUCTO
-0.066645	...	-0.001349	0.073165	0.251641	0.141256	0.095535	-0.151889	0.032462	0.018510	DESARROLLO	SAP
-0.028005	...	-0.001009	0.032687	0.111105	0.063446	0.042913	-0.066395	0.012827	0.009460	SISTEMAS	SAP
-0.038450	...	-0.002422	0.041512	0.150745	0.084061	0.057598	-0.093744	0.020908	0.008188	DESARROLLO	SAP
-0.052140	...	-0.001047	0.058881	0.198536	0.111572	0.077108	-0.120346	0.025972	0.015492	DESARROLLO	SAP
-0.043178	...	-0.001394	0.048854	0.165449	0.095447	0.062699	-0.099075	0.020219	0.010966	CONSULTORÍA	SAP
...	...	...	...	...	...	...	...	...	...	...	...
-0.040377	...	-0.007583	0.053698	0.182225	0.102090	0.063857	-0.105923	0.030362	0.009548	CONSULTORÍA	SAP
-0.039131	...	-0.001244	0.043961	0.151926	0.087512	0.058924	-0.090764	0.018372	0.011506	CONSULTORÍA	SAP
-0.047899	...	0.000317	0.055091	0.184642	0.103949	0.072257	-0.111341	0.026854	0.013850	CONSULTORÍA	SAP
-0.041065	...	0.000072	0.047499	0.163877	0.090311	0.064084	-0.095829	0.025220	0.009451	SISTEMAS	SAP
-0.036116	...	0.001333	0.044486	0.149836	0.085537	0.057713	-0.088730	0.018401	0.009731	CONSULTORÍA	SAP

Figura 4.11.: Resultados de Word2Vec

4.3.3. FastText

FastText [23] es una biblioteca, similar a Word2Vec, utilizada para la representación de textos como vectores numéricos. Difiere de Word2Vec en que introduce la representación de n-gramas o subpalabras, lo que permite manejar palabras desconocidas o raras y capturar mejor el significado de las palabras compuestas.

Utiliza una arquitectura de redes neuronales para aprender las representaciones de palabras y textos. Entrena un modelo que permite aprender representaciones vectoriales tanto para palabras individuales como para conjuntos de palabras, como frases o documentos completos, lo cual le permite representar información sobre el contexto del documento.

Para la implementación de esta técnica se ha utilizado la función *FastText()* de la librería *gensim,* y se ha seguido un proceso similar al utilizado con Word2Vec, para representar cada texto como un vector de tamaño 100 que sea el resultado de la media de los vectores de sus palabras.

```
# Este código sigue un proceso similar al aplicado con Word2Vec
# pero utilizando FastText, que en esencia proporciona resultados
# similares pero con una representación a priori más precisa

# Tokenización y creación de una lista de palabras
tokenized_texts = [text.lower().split() for text in incidencias['texto_unificado_lematizado']]

# Entrenamiento de FastText
model = FastText(sentences=tokenized_texts, vector_size=100, min_count=1)

# Representación vectorial de los textos
text_vectors = []
for text in tokenized_texts:
    vectors = [model.wv[word] for word in text if word in model.wv]
    if vectors:
        text_vector = np.mean(vectors, axis=0)
        text_vectors.append(text_vector)
    else:
        text_vectors.append(np.zeros(100))

# Se construye el DF
incidencias_fasttext = pd.DataFrame(text_vectors)
incidencias_fasttext['Categoría'] = incidencias['Categoría']
incidencias_fasttext['PRODUCTO'] = incidencias['PRODUCTO']

incidencias_fasttext
```

Figura 4.12.: Implementación de FastText

8	9	...	92	93	94	95	96	97	98	99	Categoría	PRODUCTO
0.426928	-0.326251	...	0.292383	0.279224	-0.918714	0.461779	0.309632	0.039193	-0.544232	-0.859556	DESARROLLO	SAP
0.345300	-0.262664	...	0.235510	0.225837	-0.742844	0.373457	0.249843	0.031234	-0.439579	-0.692622	SISTEMAS	SAP
0.449969	-0.343736	...	0.307705	0.292605	-0.968172	0.487062	0.326920	0.042084	-0.572855	-0.904470	DESARROLLO	SAP
0.490602	-0.374372	...	0.335201	0.321160	-1.056568	0.531070	0.355413	0.045717	-0.624764	-0.985600	DESARROLLO	SAP
0.423727	-0.323214	...	0.289556	0.276868	-0.912699	0.459022	0.306811	0.038471	-0.539789	-0.851270	CONSULTORÍA	SAP
...	...	...	...	...	...	...	...	...	...	...	...	...
0.469008	-0.356424	...	0.321845	0.307302	-1.013303	0.508846	0.339454	0.044741	-0.597718	-0.943726	CONSULTORÍA	SAP
0.453514	-0.345659	...	0.309609	0.296355	-0.978066	0.491483	0.328329	0.042132	-0.577529	-0.911123	CONSULTORÍA	SAP
0.409712	-0.312298	...	0.279995	0.268794	-0.883435	0.444275	0.296905	0.037307	-0.522288	-0.823878	CONSULTORÍA	SAP
0.383365	-0.292685	...	0.261364	0.250897	-0.826687	0.414025	0.276885	0.034296	-0.487236	-0.771589	SISTEMAS	SAP
0.398408	-0.304157	...	0.272471	0.260961	-0.858972	0.432085	0.288922	0.036784	-0.508001	-0.801052	CONSULTORÍA	SAP

Figura 4.13.: Resultados de FastText

Tras la creación de los 3 conjuntos de datos utilizando cada una de estas técnicas de vectorización, se procederá a utilizarlos todos para construir distintos modelos clasificadores, de forma que en última instancia se pueda determinar cuál de estas técnicas ha obtenido mejores resultados. Pero en primer lugar, ahora que se tienen los datos en formato numérico, es posible aplicar el rebalanceo comentado anteriormente.

4.3.4. Rebalanceo de clases

El rebalanceo de clases consiste en llevar a cabo acciones para corregir el desequilibrio en la proporción de las diferentes etiquetas en las que se clasifica un conjunto de datos. Existen varios enfoques para este proceso:

- Submuestreo de la clase mayoritaria. Se eliminan de forma aleatoria instancias de la etiqueta con mayor frecuencia, de forma que se iguale con la clase minoritaria.
- Sobremuestreo de la clase minoritaria. Se generan nuevas instancias de las clases con menor frecuencia para igualarlas con la clase mayoritaria. Estas nuevas instancias pueden ser artificiales o duplicadas de las ya existentes.
- Técnicas híbridas. Combinan los dos anteriores.

Debido a que el conjunto de datos no es demasiado grande, no es aconsejable eliminar entradas, por lo que se ha decidido adoptar una técnica de sobremuestreo aleatorio de clases llamada *SMOTE* [24].

Esta técnica genera nuevas muestras sintéticas mediante la interpolación de características de los casos existentes. A grandes rasgos, su proceso es:

- Para cada ejemplo en la clase minoritaria, SMOTE selecciona k ejemplos cercanos aleatoriamente de la misma clase (vecinos más cercanos).
- Se generan nuevos ejemplos sintéticos en la vecindad de los ejemplos seleccionados. Esto se hace tomando una combinación lineal de las características de un ejemplo seleccionado y uno de sus vecinos más cercanos, y agregando un factor aleatorio.
- Los nuevos ejemplos sintéticos se agregan al conjunto de datos, lo que aumenta el número de muestras de la clase minoritaria.
- El proceso se repite hasta que se alcance un nivel deseado de equilibrio entre las clases.

La implementación de esta técnica se ha llevado a cabo utilizando la función *SMOTE()* de la librería *imblearn*, que se ha aplicado sobre los 3 dataframes generados con las diferentes técnicas de vectorización y para cada una de las variables a predecir (Categoría y Producto).

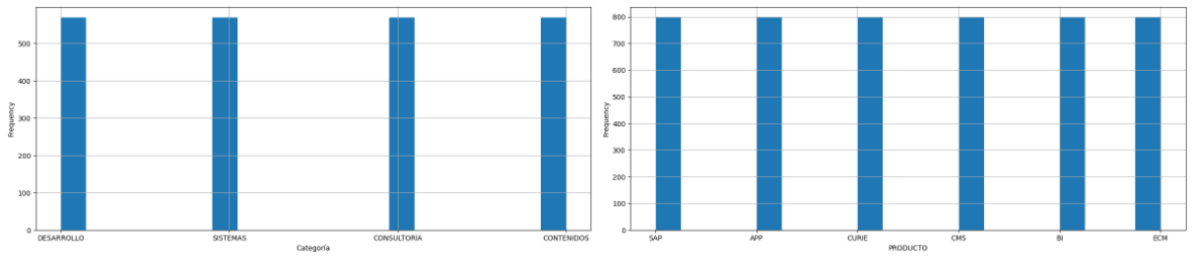


Figura 4.14.: Clases balanceadas tras la aplicación de SMOTE

Al finalizar este proceso, se tendrán 6 conjuntos de datos para utilizar en el entrenamiento de modelos posterior:

- Vectorización: TF-IDF. Variable balanceada: Categoría
- Vectorización: TF-IDF. Variable balanceada: Producto
- Vectorización: Word2Vec. Variable balanceada: Categoría
- Vectorización: Word2Vec. Variable balanceada: Producto
- Vectorización: FastText. Variable balanceada: Categoría
- Vectorización: FastText. Variable balanceada: Producto

4.4. Modelado y entrenamiento

En esta sección se emplearán diversos algoritmos para construir y entrenar múltiples clasificadores a partir de los datos disponibles, y que posteriormente serán evaluados para determinar si cumplen con los objetivos del proyecto y, en el caso de utilizar múltiples algoritmos, realizar una comparativa y decidir cuál de ellos se adapta mejor al problema actual.

Además, para ajustar al máximo estos modelos, se recurrirá a la técnica *GridSearch* [25], que consiste en explorar exhaustivamente diferentes combinaciones de hiperparámetros [26] para encontrar la configuración óptima que maximice el rendimiento del modelo.

Este algoritmo utiliza validación cruzada sobre el conjunto de datos con todas las combinaciones posibles dentro de un conjunto de parámetros que se le indique, y retorna el conjunto de parámetros que obtiene mejores resultados, es decir, los parámetros óptimos.

En este caso concreto, se ha dividido de forma aleatoria cada uno de los 6 conjuntos de datos previamente definidos en 2 subconjuntos de entrenamiento y de validación, con una proporciones del 80% de los datos para entrenamiento y el 20% para validación, y se han construido clasificadores utilizando 3 algoritmos diferentes:

- Support Vector Machine (SVM)
- Random Forest
- Naive Bayes

4.4.1. Support Vector Machine (SVM)

El algoritmo de Máquinas de Vectores de Soporte [27] (Support Vector Machine, SVM) es un método de aprendizaje supervisado utilizado generalmente para la clasificación y la regresión.

El objetivo de este método es hallar un hiperplano dentro de un espacio dimensional que sea capaz de separar los elementos pertenecientes a dos clases de forma óptima, es decir, maximizando la distancia entre las muestras de diferentes clases que estén más cercanas al hiperplano.

Esta delimitación entre clases puede definirse en base a múltiples funciones kernel que maximicen dicha distancia, como la función lineal, la polinómica o la de base radial o gaussiana. Estas tres han sido las utilizadas en este ejemplo práctico.

Este método es especialmente útil para manejar datos con alta dimensionalidad, y además es resistente al sobreajuste.

Cabe destacar que SVM es originalmente un algoritmo de clasificación binaria, es decir, que sólo puede separar 2 clases a la vez. Sin embargo, la función *SVC()* de la librería *sklearn*, permite la construcción de modelos SVM multiclase gracias a la técnica *One vs Rest* [28], que consiste en construir un modelo por cada clase existente, y llevar a cabo una clasificación entre dicha clase y el resto de ellas. De esta forma, al comparar resultados, se obtiene la clase real a la que pertenece la instancia.

Se ha construido un clasificador para cada conjunto de datos, utilizando GridSearch para elegir los mejores parámetros (entre ellos la mejor función kernel de entre las comentadas).

```python
# Se indica una validación cruzada estratificada con 2 pliegues para GridSearch
cv = StratifiedKFold(n_splits=2)

# Definir los parámetros a probar
parameters = {'C': [1, 10, 100], 'kernel': ['linear', 'rbf', 'poly'],
              'gamma': [0.1, 0.5, 0.9], 'degree': [2, 3]}


# Crear el clasificador con SVM
svm_classifier = SVC()

# Realizar la búsqueda de parámetros mediante GridSearchCV
grid_search = GridSearchCV(svm_classifier, parameters, cv=cv)
grid_search.fit(X_train_tfidf_categoria, y_train_tfidf_categoria)

# Obtener los mejores parámetros encontrados
best_params = grid_search.best_params_
print("Mejores parámetros:", best_params)

# Entrenar el clasificador con SVM y los mejores parámetros
clasificador_svm_tfidf_categoria = SVC(**best_params)
clasificador_svm_tfidf_categoria.fit(X_train_tfidf_categoria, y_train_tfidf_categoria)
```

```
Mejores parámetros: {'C': 10, 'degree': 2, 'gamma': 0.1, 'kernel': 'rbf'}
SVC
SVC(C=10, degree=2, gamma=0.1)
```

Figura 4.15.: Construcción de modelo SVM

4.4.2. Random Forest

El método Bosque Aleatorio [29] (Random Forest) es un algoritmo de aprendizaje automático utilizado para problemas de clasificación y regresión. Es una técnica de conjunto que combina múltiples árboles de decisión para obtener una predicción más robusta y precisa.

El concepto principal de Random Forest se basa en construir un conjunto de árboles independientes y combinar sus predicciones. Cada uno de estos árboles es entrenado con una muestra aleatoria de datos de entrenamiento y considera un subconjunto aleatorio de características, lo cual ayuda a reducir el sobreajuste y aumentar la diversidad dentro del conjunto de árboles.

El resultado de la predicción se obtiene por "votación" de todos los árboles individuales, siendo el resultado elegido aquel al que se haya llegado más veces.

Los Random Forests tienen varias ventajas, como su capacidad para manejar conjuntos de datos grandes con una alta dimensionalidad, su resistencia al sobreajuste y su capacidad para capturar relaciones no lineales y características importantes en los datos.

Este algoritmo también se ha empleado para construir un clasificador por cada conjunto de datos de los disponibles, gracias a la función *RandomForestClassifier()* de la librería *sklearn*, en combinación con la técnica GridSearch para la optimización de hiperparámetros.

```
# Definir los parámetros a probar
parameters = {'n_estimators': [50, 100, 200, 300, 500], 'max_depth': [None, 5, 10], 'min_samples_split': [2, 5, 10]}

# Crear el clasificador Random Forest
rf_classifier = RandomForestClassifier()

# Realizar la búsqueda de parámetros mediante GridSearchCV
grid_search = GridSearchCV(rf_classifier, parameters)
grid_search.fit(X_train_tfidf_categoria, y_train_tfidf_categoria)

# Obtener los mejores parámetros encontrados
best_params = grid_search.best_params_
print("Mejores parámetros:", best_params)

# Entrenar el clasificador Random Forest con los mejores parámetros
clasificador_randforest_tfidf_categoria = RandomForestClassifier(**best_params)
clasificador_randforest_tfidf_categoria.fit(X_train_tfidf_categoria, y_train_tfidf_categoria)
```

```
Mejores parámetros: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 300}
RandomForestClassifier(n_estimators=300)
```

Figura 4.16.: Construcción de modelo Random Forest

4.4.3. Naive Bayes

Naive Bayes [30] es un algoritmo de clasificación supervisada basado en el Teorema de Bayes [31].

Este tipo de modelos se llaman “Naive” o “Inocentes”, debido a que asumen que las variables predictoras son independientes entre sí , en otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica.

Utiliza el Teorema de Bayes para calcular la probabilidad de que una instancia pertenezca a una determinada clase dada su combinación de atributos.

Esta técnica es rápida y eficiente en términos de tiempo de entrenamiento y predicción, especialmente en conjuntos de datos grandes, debido en gran parte a su simplicidad.

Se ha implementado gracias a la función *GaussianNB(),* de *sklearn,* y dado que este método no presenta hiperparámetros que ajustar, no ha sido necesario el uso de GridSearch.

```
# Crear el clasificador Naive Bayes
clasificador_naivebayes_tfidf_categoria = GaussianNB()
clasificador_naivebayes_tfidf_categoria.fit(X_train_tfidf_categoria, y_train_tfidf_categoria)
```

```
GaussianNB
GaussianNB()
```

Figura 4.17.: Construcción de modelo Naive Bayes

Una vez construidos todos los modelos, se puede proceder con la etapa de evaluación de clasificadores

4.5. Evaluación de los clasificadores

Para el proceso de validación de clasificadores, se mide el rendimiento de los mismos en base a su capacidad para realizar predicciones acertadas sobre un conjunto de datos de prueba.

En esta evaluación se estudian diferentes métricas dadas por los modelos:

- Precisión. Es la proporción de instancias correctamente clasificadas como positivas. No confundir con *accuracy*.

$$Precisión = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ positivos}$$

- Recall o sensibilidad. Proporción de instancias correctamente clasificadas como positivas en relación a todas las instancias realmente positivas.

$$Recall = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos}$$

- F1-score. Es una medida global que combina la precisión y el recall. Calcula la media armónica entre ambas, en lugar de la media aritmética, para evitar que el F1 Score sea dominado por valores muy pequeños.

$$F1 - Score = 2\ x\ \frac{Precisión\ x\ Recall}{Precisión + Recall}$$

- Accuracy. Proporción de instancias correctamente clasificadas con respecto al total.

$$Accuracy = \frac{Verdaderos\ positivos + Verdaderos\ negativos}{Verdaderos\ positivos + Verdaderos\ negativos + Falsos\ positivos + Falsos\ negativos}$$

- Matriz de confusión. Es una tabla que muestra el desempeño de un modelo de clasificación al predecir las etiquetas de las muestras. En su forma más básica, tiene filas que representan las etiquetas reales y columnas que representan las etiquetas predichas. Los valores en la matriz indican la cantidad de muestras que fueron clasificadas correctamente o incorrectamente según la predicción del modelo.

En este caso concreto, se han validado todos los modelos construidos (18 en total: 6 conjuntos de datos por 3 algoritmos utilizados) con respecto al conjunto del 20% de los datos destinado a los tests y se han generado todas las medidas anteriormente mencionadas. Para evitar la saturación de contenido, se mostrará únicamente un ejemplo para mostrar la obtención de las métricas previamente comentadas, y se pasará a la comparación entre los clasificadores.

```python
# Realizar predicciones en los datos
y_pred = clasificador_svm_tfidf_categoria.predict(X_test_tfidf_categoria)

# Evaluar el rendimiento del clasificador
f1score_svm_tfidf_categoria = f1_score(y_test_tfidf_categoria, y_pred, average='weighted')
report = classification_report(y_test_tfidf_categoria, y_pred)

# Imprimir los resultados
print("Reporte de clasificación:")
print(report)

cm = confusion_matrix(y_test_tfidf_categoria, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# Configuraciones adicionales
plt.title('Matriz de Confusión')
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Real')
plt.show()
```

Figura 4.18.: Código estándar para la predicción y generación de métricas

```
Reporte de clasificación:
              precision    recall  f1-score   support

  CONSULTORÍA       0.94      0.92      0.93       129
   CONTENIDOS       1.00      1.00      1.00       105
   DESARROLLO       0.89      0.91      0.90       116
     SISTEMAS       0.97      0.97      0.97       106

     accuracy                           0.95       456
    macro avg       0.95      0.95      0.95       456
 weighted avg       0.95      0.95      0.95       456
```

Figura 4.19.: Muestra de métricas de evaluación de SVM

Los resultados obtenidos por el algoritmo SVM han sido bastante buenos: se observan una precisión y un *recall* medios de un 95%, siendo la categoría 'Desarrollo' la que presenta un mayor número de falsos positivos, con solo un 89% de precisión, y también la que más falsos negativos ha mostrado, con un 91% de recall, reflejando así el menor valor de *f1-score* con un 90%. La exactitud o *accuracy* en este caso concreto es de un 95%, lo cual se considera un resultado bastante aceptable.

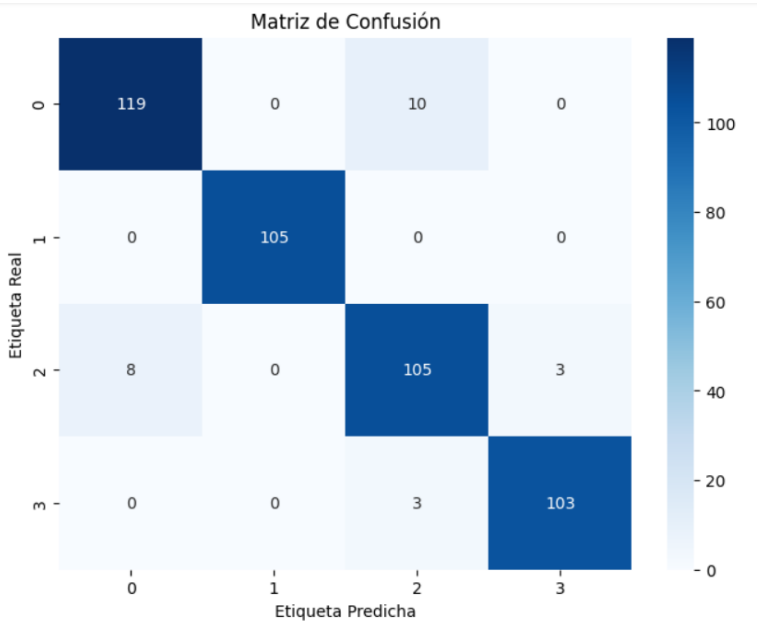


Figura 4.20.: Matriz de confusión SVM

La matriz de confusión muestra de una forma gráfica y visual la proporción de instancias mal etiquetadas. Se aprecia claramente que la categoría 1 (‘Contenidos’) no ha sido etiquetada de forma incorrecta, recordando su precisión y su *recall* del 100%. Por otra parte, podría decirse que la categoría 2 (‘Desarrollo’) es la que peor se ha clasificado, ya que muestra una ligera tendencia a confundirse con la categoría 0 (‘Consultoría’), pero aún así, se mantiene una proporción bastante buena de clasificaciones correctas.

Pasando a la comparación entre algoritmos, se ha construido una tabla que contiene el valor de la métrica que se ha definido inicialmente para evaluar este problema (*f1-score*) para cada uno de los algoritmos de clasificación y cada uno de los conjuntos de datos.

Conjunto de Datos	SVM	Random Forest	Naive Bayes
TFIDF – Categoría	0.947439	0.913431	0.904209
TFIDF – Producto	0.997908	0.991655	0.993714
Word2Vec – Categoría	0.706706	0.862908	0.420757
Word2Vec – Producto	0.846136	0.966326	0.263323
FastText – Categoría	0.783515	0.759848	0.303682
FastText – Producto	0.924852	0.871563	0.317996

Tabla 4.1.: Comparativa de resultados entre algoritmos

De esta tabla pueden extraerse múltiples conclusiones:

- En términos generales, se obtienen mejores resultados a la hora de predecir el producto al que se relaciona la incidencia, en lugar de su categoría. Probablemente el proceso de rebalanceo con SMOTE haya jugado un papel importante en esta diferenciación.

- De entre las técnicas de vectorización aplicadas, TF-IDF es la que ha obtenido claramente mejores resultados. Claro indicador de que, para el conjunto de datos del que se dispone, TF-IDF ha sido capaz de extraer una mayor información que Word2Vec y FastText. Probablemente, con un dataset más extenso, los resultados habrían sido distintos.

- SVM es sin duda el clasificador que se debe elegir en esta comparativa, llegando a obtener un 99.79% de *f1-score.* Aunque este valor depende del conjunto de datos elegido y del proceso por el que se le haga pasar, ya que en términos generales, Random Forest obtiene unos mejores resultados de media, pero en el caso concreto de la predicción de Productos con TF-IDF, se ve superado por SVM.

En resumen, pese a no disponer de una línea base con la que comparar los resultados, resulta evidente que el 99.79% de *f1-score* obtenido supera con creces cualquier rendimiento humano. Por tanto, en el caso de poner en producción este proyecto, se haría utilizando la técnica de vectorización TF-IDF y el algoritmo de clasificación SVM para clasificar las incidencias de la empresa en base al producto al que corresponden.

Capítulo 5. Resultados

En este proyecto de clasificación de incidencias se han puesto en práctica las recomendaciones y la estructura de un proyecto de Aprendizaje Automático estándar, donde se han llevado a cabo la mayoría de etapas de su ciclo de vida habitual.

En primer lugar, se ha definido el problema a resolver, de forma concisa y detallada, donde se han planteado claramente sus objetivos y las métricas de evaluación requeridas y el arquetipo del proyecto con el que se relaciona.

Posteriormente, dado que el paso de recolección y etiquetado de datos ha sido realizado por iTop S.L., se ha avanzado directamente al análisis y preprocesado de los mismos, donde ha resultado necesario llevar a cabo acciones como limpieza de los textos, eliminación de valores nulos, lematización de palabras y rebalanceo de clases. Este último paso ha tenido que retrasarse debido a la representación original de los datos.

En el nivel de extracción de características, se han empleado tres algoritmos de vectorización diferentes con el fin de realizar una comparativa entre ellos en base a los resultados obtenidos: TF-IDF, Word2Vec y FastText, todos ellos orientados a la extracción de información de los textos y su representación numérica. Tras esto ha sido posible ejecutar el rebalanceo.

La construcción de los modelos se ha realizado sobre seis conjuntos de datos distintos, generados por los diferentes algoritmos de vectorización y los diferentes rebalanceos aplicados. Tras aplicarse tres algoritmos de clasificación diferentes con el fin de comparar sus resultados, y de entrenar estos modelos sobre un conjunto de datos de entrenamiento, finalmente se ha obtenido un total de 18 clasificadores a evaluar.

Durante la evaluación de estos modelos, se han generado y comentado diversas métricas que se utilizan para el estudio de los resultados. Sin embargo, los esfuerzos se han centrado en la métrica indicada inicialmente como crítica para este problema: la *f1-score*. Utilizando este valor para analizar una comparativa, se ha determinado que el mejor proceso a seguir es utilizar un algoritmo de vectorización TF-IDF, en combinación con un sobremuestreo de los datos orientado a la clase ‘Producto’ y aplicado a un algoritmo de clasificación SVM, obteniendo un resultado considerablemente bueno, 99.79%.

Una vez llegados a este punto, y dado que se trata simplemente de un ejemplo práctico, no ha correspondido avanzar a las fases de puesta en producción y mantenimiento, aunque en un caso real, sería tarea de la empresa el llevarlas a cabo (en este caso, iTop S.L.).

A modo de resumen, el proceso seguido puede explicarse conceptualmente con el siguiente esquema.

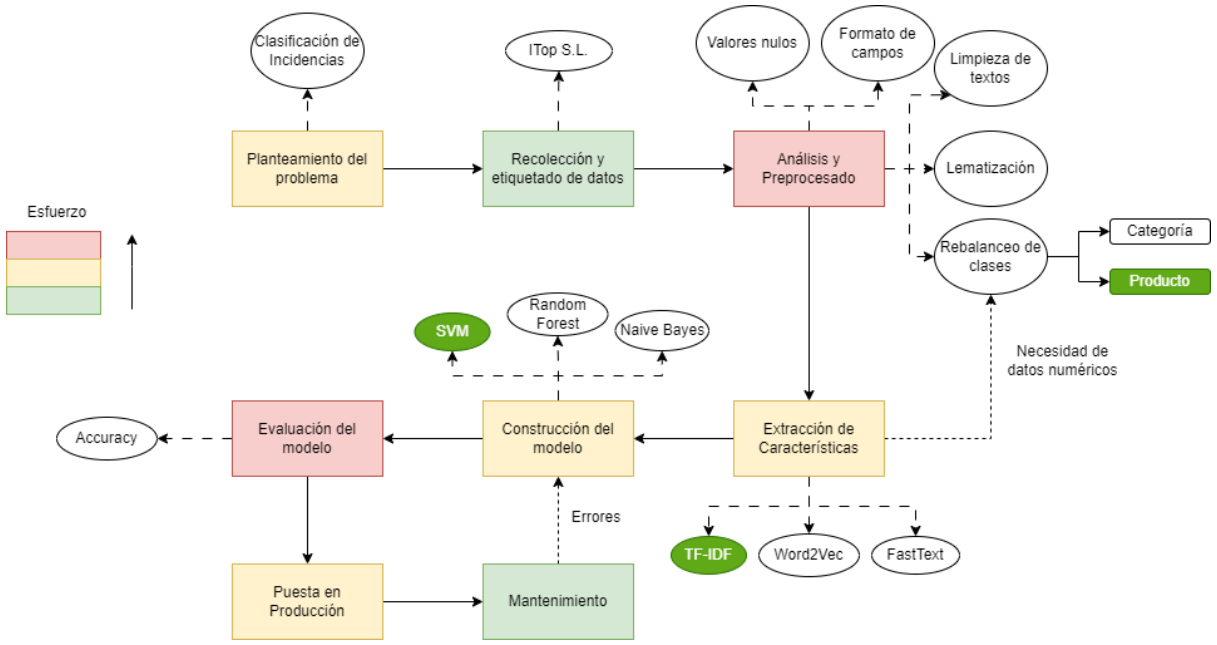


Figura 5.1.: Ciclo de vida aplicado a la Clasificación de Incidencias

Capítulo 6. Presupuesto

En la Tabla 6.1 se detallan las tareas realizadas y el tiempo invertido en cada una en horas, además de los recursos necesarios para llevar a cabo este proyecto, todo ello acompañado de su precio correspondiente.

El salario medio de un Ingeniero Informático es de unos 36.500,00 euros brutos anuales, alrededor de 3.000,00 euros brutos mensuales [32]. Según esto, y teniendo en cuenta que un mes tiene 20 días laborales y suponiendo una jornada de 8 horas, el coste de cada hora de trabajo es de 19,00 euros brutos.

Todo esto, aplicado a las horas de trabajo dedicadas a cada fase del proyecto, y sumado a los recursos necesarios para su realización, ofrece el presupuesto total en bruto del proyecto.

Descripción	**Cantidad (horas)**	**Precio (€)**	**Total (€)**
Planificación del proyecto	10	19,00	190,00
Documentación y estudio	20	19,00	380,00
Planteamiento del problema	10	19,00	190,00
Análisis y preprocesado	20	19,00	380,00
Extracción de características	15	19,00	285,00
Modelado y entrenamiento	25	19,00	475,00
Evaluación de resultados	20	19,00	380,00
Redacción de la memoria	20	19,00	380,00
Ordenador portátil	1	599.00	599,00
		TOTAL:	3.259,00

Tabla 6.1.: Presupuesto del proyecto

Capítulo 7. Conclusiones y líneas futuras

El aprendizaje automático ha revolucionado enormemente la forma en que se aborda el análisis de datos y la resolución de problemas. Esta disciplina de la inteligencia artificial ha permitido construir modelos y algoritmos que aprenden de los datos, identifican patrones ocultos y realizan predicciones precisas en una amplia gama de aplicaciones.

En cambio, el éxito de estos modelos depende en gran medida de la calidad de los datos, la selección adecuada de algoritmos y la correcta configuración de los hiperparámetros.

En este Trabajo de Fin de Máster, se ha estudiado en profundidad la estructura y las consideraciones a tener en cuenta en un proyecto de ML y se ha desarrollado un proyecto real con aplicaciones reales en empresas a día de hoy.

Pese al buen resultado obtenido en la clasificación de incidencias llevada a cabo, este modelo no está listo para su implementación en el mundo real, dado que el conjunto de datos inicial, con aproximadamente mil entradas, no es suficiente para considerar que el clasificador haya sido entrenado en condiciones.

Sin embargo, el código realizado ha permitido mostrar una aplicación práctica del ciclo de vida típico de los proyectos de aprendizaje automático, donde ha quedado patente la importancia del tratamiento de los datos, así como la correcta elección de los algoritmos a utilizar. No obstante, los métodos empleados y comparados entre sí representan una pequeña parte de todas las técnicas aplicables en el machine learning, por lo que siempre será importante considerar el abordaje más adecuado para el contexto del problema.

Por tanto, si hubiera que definir alguna línea futura para el proyecto de clasificación de incidencias, esta pasaría sin duda por utilizar un conjunto de datos muchísimo más amplio, lo cual probablemente requeriría de afinar el preprocesado de los datos e incluso investigar otras técnicas de extracción de características u otros algoritmos clasificadores que pudieran mejorar los resultados.

A medida que la tecnología continúa avanzando, es probable que el aprendizaje automático desempeñe un papel aún más destacado en la sociedad, impulsando la innovación y permitiendo soluciones más inteligentes y eficientes. Sin embargo, es esencial abordar los desafíos éticos y asegurar que su implementación se realice de manera responsable y beneficiosa para la humanidad.

Capítulo 8. Summary and Conclusions

Machine learning has greatly revolutionized the way data analysis and problem-solving are approached. This discipline of artificial intelligence has enabled the construction of models and algorithms that learn from data, identify hidden patterns, and make accurate predictions in a wide range of applications.

However, the success of these models highly depends on the quality of the data, the correct choice of algorithms and the right configuration of hyperparameters.

In this project, the structure and considerations in a machine learning project have been thoroughly studied, and a real project with practical applications in today's businesses has been developed.

Despite the good results achieved in the ticket classification carried out, this model is not ready for real-world implementation as the initial dataset, with approximately a thousand entries, is not sufficient to consider the classifier trained under suitable conditions.

Nevertheless, the implemented code has demonstrated a practical application of the typical lifecycle of machine learning projects, highlighting the importance of data preprocessing and the proper selection of algorithms. However, the methods used and compared represent only a small portion of the many techniques applicable in machine learning, emphasizing the ongoing importance of considering the most suitable approach for the problem context.

Therefore, if there were to define a future direction for the incident classification project, it would undoubtedly involve using a much larger dataset. This would likely require fine-tuning the data preprocessing and even exploring other feature extraction techniques or classifiers that could improve the results.

As technology continues to advance, it is likely that machine learning will play an even more prominent role in society, driving innovation and enabling smarter and more efficient solutions. However, it is essential to address the ethical challenges and ensure that its implementation is done responsibly and for the benefit of humanity.

Bibliografía

[1] T. rédac, «Machine Learning: definición, funcionamiento, usos», *Formation Data Science | DataScientest.com*, 13 de diciembre de 2021. https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos (accedido 30 de junio de 2023).

[2] «Big Data: ¿En qué consiste? Su importancia, desafíos y gobernabilidad». https://www.powerdata.es/big-data (accedido 30 de junio de 2023).

[3] «ChatGPT». https://openai.com/chatgpt (accedido 30 de junio de 2023).

[4] «Sistemas de recomendación | Qué son, tipos y ejemplos». https://www.grapheverywhere.com/sistemas-de-recomendacion-que-son-tipos-y-ejemplos/ (accedido 30 de junio de 2023).

[5] «Unidad de procesamiento gráfico», *Wikipedia, la enciclopedia libre*. 22 de junio de 2023. Accedido: 30 de junio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_de_procesamiento_gr%C3%A1fico&oldid=152012831

[6] «Unidad de procesamiento tensorial», *Wikipedia, la enciclopedia libre*. 2 de junio de 2023. Accedido: 30 de junio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_de_procesamiento_tensorial&oldid=151591874

[7] «¿Qué es TensorFlow y para qué sirve?», *¿Qué es TensorFlow y para qué sirve?* https://www.incentro.com/es-ES/blog/que-es-tensorflow (accedido 30 de junio de 2023).

[8] «¿Qué es el procesamiento del lenguaje natural? | IBM». https://www.ibm.com/es-es/topics/natural-language-processing (accedido 30 de junio de 2023).

[9] «Setting up Machine Learning Projects». https://fall2019.fullstackdeeplearning.com/course-content/setting-up-machine-learning-projects (accedido 30 de junio de 2023).

[10] «Qué es un arquetipo | Blog | Hosting Plus España», *Hosting Plus*, 8 de diciembre de 2021. https://www.hostingplus.com.es/blog/que-es-un-arquetipo/ (accedido 3 de julio de 2023).

[11] «9 - Ciclo de vida de un proyecto de Machine Learning», *Codificando Bits*. https://www.codificandobits.com/curso/introduccion-machine-learning/9-ciclo-de-vida-proyecto-machine-learning/ (accedido 1 de julio de 2023).

[12] «Lematización», *Wikipedia, la enciclopedia libre*. 11 de enero de 2022. Accedido: 1 de julio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Lematizaci%C3%B3n&oldid=140877099

[13] «¿Qué son las Stop Words y para qué sirven? – BlackBeast», *Blackbeast.pro*. https://blackbeast.pro/diccionario/stop-words/ (accedido 1 de julio de 2023).

[14] «Google Colaboratory». https://colab.research.google.com/?hl=es (accedido 1 de julio de 2023).

[15] «Welcome to Python.org», *Python.org*, 1 de julio de 2023. https://www.python.org/ (accedido 1 de julio de 2023).

[16] «Plataforma de almacenamiento personal en la nube y uso compartido de archivos - Google». https://www.google.com/intl/es/drive/ (accedido 1 de julio de 2023)

[17] «Documentos de Google: editor de documentos online | Google Workspace». https://www.facebook.com/GoogleDocs/ (accedido 1 de julio de 2023).

[18] Itop, «Itop». https://www.itop.es/ (accedido 1 de julio de 2023).

[19] «Cuaderno de Jupyter con el Proyecto Clasificador de Tickets». https://colab.research.google.com/drive/1Srti44Gxi9i2P50_7KoZoQ4KnsMOxYnS?usp=sharing (accedido 1 de julio de 2023).

[20] «spaCy · Industrial-strength Natural Language Processing in Python». https://spacy.io/ (accedido 1 de julio de 2023).

[21] «Tf-idf», *Wikipedia, la enciclopedia libre*. 11 de diciembre de 2019. Accedido: 2 de julio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Tf-idf&oldid=121943707

[22] T. Filzinger, «Word2vec: análisis de textos mediante incrustación de palabras», *Konfuzio*, 29 de mayo de 2023. https://konfuzio.com/es/wordtovec/ (accedido 2 de julio de 2023).

[23] https://www.facebook.com/grokkeepcoding, «¿Qué es FastText y cómo funciona? | KeepCoding Bootcamps», 27 de febrero de 2023. https://keepcoding.io/blog/que-es-fasttext-y-como-funciona/ (accedido 2 de julio de 2023).

[24] likebupt, «SMOTE - Azure Machine Learning», 1 de junio de 2023. https://learn.microsoft.com/es-es/azure/machine-learning/component-reference/smote (accedido 2 de julio de 2023).

[25] https://www.facebook.com/grokkeepcoding, «¿Qué es GridSearchCV? | KeepCoding Bootcamps», 30 de noviembre de 2022. https://keepcoding.io/blog/que-es-gridsearchcv/ (accedido 3 de julio de 2023).

[26] D. Rodríguez, «¿Cuál es la diferencia entre parámetro e hiperparámetro?», *Analytics Lane*, 16 de diciembre de 2019. https://www.analyticslane.com/2019/12/16/cual-es-la-diferencia-entre-parametro-e-hiperparametro/ (accedido 3 de julio de 2023).

[27] «Support Vector Machine (SVM)». https://es.mathworks.com/discovery/support-vector-machine.html (accedido 4 de julio de 2023).

[28] J. Brownlee, «One-vs-Rest and One-vs-One for Multi-Class Classification», *MachineLearningMastery.com*, 12 de abril de 2020. https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/ (accedido 4 de julio de 2023).

[29] T. rédac, «Random Forest: Bosque aleatorio. Definición y funcionamiento», *Formation Data Science | DataScientest.com*, 25 de enero de 2022. https://datascientest.com/es/random-forest-bosque-aleatorio-definicion-y-funcionamiento (accedido 4 de julio de 2023).

[30] V. Roman, «Algoritmos Naive Bayes: Fudamentos e Implementación», *Ciencia y Datos*, 29 de abril de 2019. https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f (accedido 4 de julio de 2023).

[31] «Teorema de Bayes», *Wikipedia, la enciclopedia libre*. 13 de junio de 2023. Accedido: 4 de julio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Teorema_de_Bayes&oldid=151826260

[32] «¿Cuánto Cobra un Ingeniero Informático? (Sueldo 2023) | Jobted.es». https://www.jobted.es/salario/ingeniero-inform%C3%A1tico (accedido 5 de julio de 2023).