ESCUELA DE INGENIERÍA DE
TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Aplicación de algoritmos de Machine Learning para la clasificación documental

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Telemática

Autor: Sr. Erik Martrus Guillén

Tutores: Sr. Álvaro Suárez Sarmiento

Sra. Elsa María Macías López

Fecha: Julio 2022

Índice

1. Introducción

En este capítulo se presenta la idea general del ámbito (e importancia) en el que se enmarca el sistema desarrollado y su entorno, los objetivos perseguidos, algunos conceptos básicos relacionados con las tecnologías usadas, la estructura de la memoria y un análisis de la memoria.

1.1 Antecedentes

Muchos de los servicios de las administraciones públicas que implican recepción de documentación por parte ya sea de los ciudadanos, a través de reclamaciones, solicitudes o de cualquier otro tipo de entidad ya sea gubernamental o no, genera una gran cantidad de datos que han de ser procesados y examinados con detenimiento, para así proceder a su clasificación o envío a otra entidad.

Gran parte de ese procesado implica una pérdida de tiempo y en algunos casos que conllevan la desestimación de las solicitudes, que por ejemplo no llegan a tiempo dentro de unos plazos establecidos. Por ello para agilizar este proceso, nace la necesidad de implementar un servicio que reduzca este tiempo y esta carga de trabajo que retrasa los procesos. Para lo cual decidimos a través del aprendizaje máquina (*Machine Learning* (*ML*)), clasificar los diferentes documentos de entrada citados anteriormente, haciendo uso de vectores descriptores (que tienen más valor significativo), obtenidos de herramientas como el *Optical Character Recognition* (*OCR*) o similares. El OCR es una de las áreas más antiguas dentro del reconocimiento de patrones y el procesado de imágenes y es una técnica para la digitalización de textos. El OCR es referido a menudo como la alternativa más rápida, económica y segura para la entrada automática del contenido de los documentos impresos en papel a soportes electrónicos. El OCR es una tecnología transversal, aplicable en distintos ámbitos y sectores para la digitalización de formularios, documentos administrativos, informes, facturas... ya que las ventajas que ofrece son comunes para todos ellos. Por ejemplo, en el sector de la cultura, en el ámbito de la preservación del patrimonio, el OCR se aplica principalmente en los procesos de digitalización de documentos históricos que se encuentran en soporte papel. Al permitir extraer el texto de documentos digitalizados pueden realizarse búsquedas en todo el documento y en el proceso de creación de los metadatos. El OCR, asimismo, se puede utilizar para generar índices de palabras clave del texto reconocido de forma automática (banco de datos).

Nos centramos en estudiar las herramientas más populares de software de OCR: *Tesseract OCR* [1], *OCR.SPACE* [2], *ABBYY Fine Reader OCR* [3], *ReadIRIS Pro* [4] y *Nuance OmniPage* [5] que son de las más precisas y productivas para el reconocimiento de

textos y conversión de documentos. Por otro lado, existen también distintas *Application Programming Interface* (*API*) para añadir la capacidad de digitalizar documentos al lenguaje para generar un software de OCR.

Una vez se disponga de ese banco de datos se realiza el entrenamiento mediante *ML*. El ML permite a los computadores aprender de manera autónoma a partir de distintas fuentes de información. Se utiliza para diferentes aplicaciones tales como motores de búsquedas, reconocimiento de voz o de escritura y muchas otras opciones.

Actualmente la mayoría de las industrias que trabajan con grandes cantidades de datos han reconocido el valor de la tecnología del ML. Obteniendo información de estos datos (a menudo en tiempo real), las organizaciones pueden trabajar de manera más eficiente o lograr una ventaja sobre sus competidores.

Entre los sectores más destacados donde se utiliza ML podemos mencionar: servicios financieros, gobierno, atención a la salud, marketing y ventas, transporte…

Dos de los métodos de ML más ampliamente adoptados son aprendizaje supervisado y aprendizaje no supervisado, pero existen también otros métodos [6 y 7]:

- Los algoritmos de aprendizaje supervisado son entrenados utilizando ejemplos etiquetados, como una entrada mediante la cual se conoce el resultado deseado. Se utiliza comúnmente en aplicaciones donde datos históricos predicen eventos futuros probables. Por ejemplo, puede anticipar cuándo es probable que transacciones con tarjetas de crédito sean fraudulentas o qué cliente de una aseguradora tiene la probabilidad de iniciar un reclamo.
- El aprendizaje no supervisado se utiliza contra datos que no tienen etiquetas históricas. No se da la *respuesta correcta* al sistema. El algoritmo debe descubrir lo que se muestra. El objetivo es explorar los datos y encontrar alguna estructura. Funciona bien con datos de transacciones.
- El aprendizaje semi-supervisado se utiliza para las mismas aplicaciones que el aprendizaje supervisado. Sin embargo, utiliza datos etiquetados y no etiquetados para entrenamiento; por lo general una pequeña cantidad de datos etiquetados con una gran cantidad de datos no etiquetados (porque los datos

no etiquetados son menos costosos y se requiere menos esfuerzo en su obtención). Este tipo de aprendizaje se puede utilizar con métodos como la clasificación, regresión y predicción.

- El aprendizaje con refuerzo se utiliza a menudo para robótica, juegos y navegación. Con el aprendizaje con refuerzo, el algoritmo descubre a través de ensayo y error qué acciones producen las mayores recompensas. Este tipo de aprendizaje tiene tres componentes principales: el agente (el que aprende o toma decisiones), el entorno (todo con lo que interactúa el agente) y acciones (lo que el agente puede hacer). El objetivo es que el agente elija acciones que maximicen la recompensa esperada en cierta cantidad de tiempo. Logrando así el objetivo mucho más rápido si se aplicara una buena política.

1.2 Objetivos

Este *Trabajo de Fin de Grado* (*TFG*) parte de la base de que en una Administración pública se reciben o generan documentos oficiales en papel que van destinados a la misma u otra administración. Posteriormente estos documentos se digitalizan, y a partir de ahí nuestro objetivo general es: el renderizado de documentos digitales para eliminar ruido, la aplicación de un OCR a los documentos renderizados, la generación de vectores descriptores a partir de los documentos anteriores y la realización de un estudio comparativo de las diferentes plataformas de ML existentes en la Nube, tales como: IBM Watson, Google AI, Amazon AWS AI o Weka para alcanzar una clasificación documental mediante las distintas plataformas de *Inteligencia Artificial* (*IA*) ya mencionadas. Un hecho destacable es que partimos de documentos digitalizados en formato *Portable Document Format* (*PDF*), obtenemos gran cantidad de vectores descriptores, entrenamos a la IA y valoramos su eficacia y la validez.

Este objetivo general se lleva a cabo mediante la consecución de los diferentes objetivos operativos detallados a continuación:

- Análisis de diferentes plataformas o las API de OCR para la obtención de los vectores. Para ello, se realiza un estudio de dichas plataformas, verificando su

compatibilidad entre los distintos sistemas operativos y de cómo lograr una correcta implementación en el lenguaje de programación establecido.

- Tras lograr el objetivo anterior, y aplicar con éxito el OCR, buscamos conseguir la generación de vectores descriptores a partir de documentos en PDF.
- Tras el objetivo anterior cumplido se realiza un estudio comparativo de las diferentes plataformas de ML, verificando si su implementación es viable en nuestro TFG y su compatibilidad con las aplicaciones en las que estamos interesados, así como lograr con el entrenamiento de la IA los resultados deseados.
- Logrado el objetivo anterior, se realiza un estudio de los resultados que nos arroja el entrenamiento de la IA buscando los mejores posibles.

1.3 Análisis preliminar del problema

Imagine que tiene un documento en papel, por ejemplo, un artículo de revista, un folleto o un contrato. Realizar un escaneo para editar esa información en *Microsoft Word* ® no es suficiente ya que todo lo que puede hacer un escáner es crear una imagen o una instantánea del documento, que es una colección de puntos negros y blancos o color, conocida como imagen de trama. Para extraer y reusar los datos de los documentos escaneados (digitalizados), imágenes de cámara o de PDF, se necesita un software de OCR que separe las letras de las imágenes, las ponga en palabras y, por consiguiente, las palabras en frases; lo que permite tener acceso y editar digitalmente el contenido del documento original.

Los sistemas OCR están formados por una combinación de hardware y software que se utiliza para convertir documentos físicos en texto legible por un computador. Se usa hardware, como un escáner óptico [1] o una placa de circuito especializada para copiar o leer texto, mientras que el software generalmente maneja el procesamiento avanzado. El software también puede aprovechar la IA [2] para implementar métodos más avanzados de reconocimiento inteligente de caracteres [3], como identificar idiomas o estilos de escritura a mano. Nosotros descartamos esta técnica porque con el OCR es suficiente para nuestros propósitos.

El proceso de OCR se usa comúnmente para pasar a una versión digital documentos legales o históricos en archivos PDF [4]. Una vez se ha realizado el proceso de OCR en esta copia electrónica [5], los usuarios pueden editar, formatear y buscar el documento como si hubiera sido creado con un procesador de textos [6].

Por tanto, el problema que resolvemos es la clasificación de documentos oficiales (en formato PDF) en base a las palabras claves, o más representativas de los mismos, que se descubren mediante el uso del OCR y la aplicación de ML. En la Figura 1.1 se muestran todos los pasos implicados en la solución de este problema.

La obtención de estas palabras claves se puede complicar debido a que algunos documentos oficiales digitalizados presentan parámetros de ruido, como son:

- Marcas de agua, en el caso de las empresas con el logo corporativo.
- Sellos de entidades oficiales.
- Firmas a mano o cualquier tipo de texto escrito a mano, nombres, apellidos, sombras, manchas...

Para eliminar estos ruidos hacemos filtrado de estos documentos, para poder así tratarlos con la mayor eficiencia posible.

Los documentos PDF provienen de un ente público, en concreto uno dependiente de *Puertos del Estado* (*Autoridad Portuaria de Santa Cruz de Tenerife*). Los documentos se corresponden a distintos departamentos de dicha entidad:

- Dominio Público.
- *Recursos Humanos* (*RRHH*).
- Secretaria General.
- Sistemas informáticos.
- Infraestructura.

Se renderizan los documentos PDF anteriores para su correcto procesado y poder tratarlos como imágenes, filtrando ruido, elevando su calidad y permitiendo la realización del OCR, obteniendo texto plano a partir de cada documento en PDF.

A continuación, obtenemos las palabras claves o más importantes del texto plano que conforman cada documento, formando una biblioteca o una fuente de datos con todas esas palabras, etiquetando cada palabra con el departamento al que corresponden para facilitar más adelante el entrenamiento que hacemos de la IA. Con esa fuente de datos de numerosas palabras hacemos su entrenamiento y su validación con la ML. En último lugar, analizamos los resultados y comprobamos su correcta finalidad.

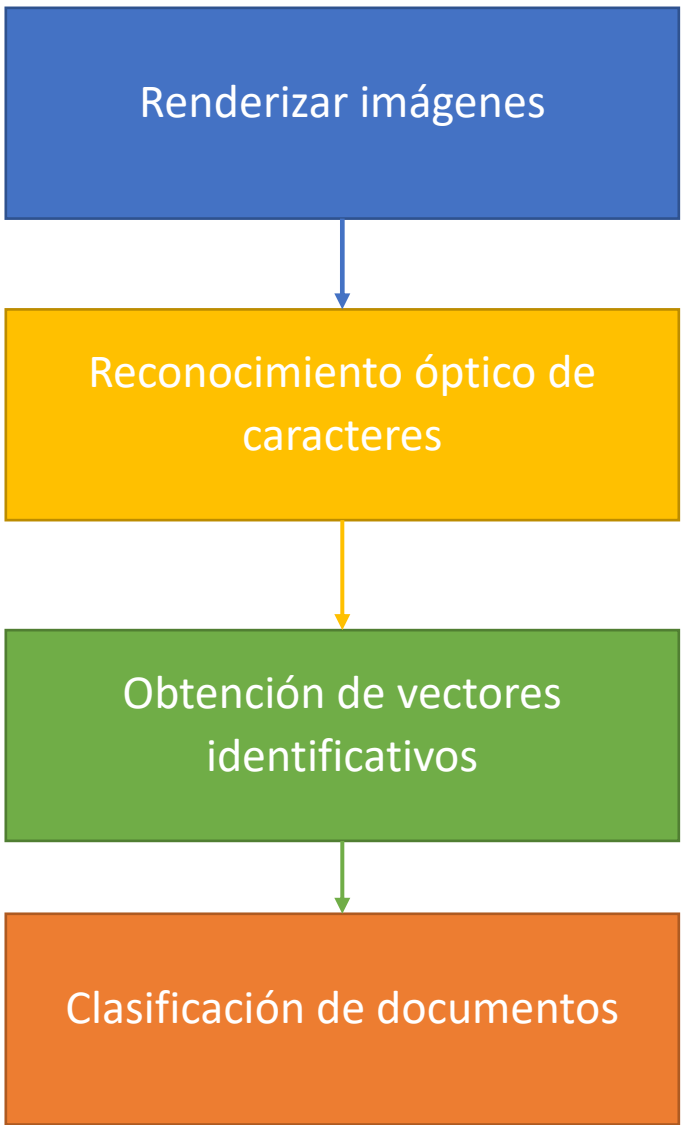


Figura 1.1. Pasos para seguir en este TFG

1.4 Estructura de la memoria

Esta memoria se estructura en diferentes capítulos.

En el capítulo 1 se explican los conceptos previos que se van a tratar en la memoria y los objetivos propuestos.

En el capítulo 2 se desarrollan el funcionamiento de las tecnologías que empleamos en nuestro proyecto y los distintos tipos que existen.

Durante el capítulo 3 se describen en detalle todas las tecnologías utilizadas para el desarrollo del proyecto.

En el capítulo 4 se explica la implementación de las tecnologías explicadas en el capítulo anterior, dentro del entorno que vamos a utilizar durante esta memoria.

En el capítulo 5 se especifican las pruebas realizadas durante el TFG y que nos han permitido verificar y validar los distintos resultados que hayamos obtenido.

Durante el capítulo 6 se concretan las conclusiones después del desarrollo de todo nuestro proyecto y se comentan posibles ampliaciones del trabajo.

Tras el capítulo anterior, tenemos un glosario y las referencias utilizadas para la elaboración de este proyecto.

A continuación, se incluye un pliego de condiciones y el presupuesto donde se expone la problemática y el coste económico de llevar a cabo todo lo expuesto en esta memoria.

Por último, en el Anexo A a D se incluye más información acerca del software utilizado y un Descargo de responsabilidades empleando la privacidad de la información con la que tratamos y protegemos la privacidad de estos.

2. Tecnologías aplicables

En este capítulo se presenta las tecnologías utilizadas en el TFG, sus distintos tipos, su funcionamiento y una valoración crítica de las mismas.

2.1 Renderizado de imágenes

Para entender cada uno de los pasos que vamos a seguir en este TFG, hemos de explicar con claridad, detenidamente, los procesos que realizamos, empezando por el renderizado de los documentos PDF. Para ello, presentamos varios conceptos previos a destacar:

- *Mallas de renderizado*: se caracteriza por ser una aproximación a una forma sin querer llegar al máximo nivel de detalle. Para logarla se establece un continuo juego de optimización de recursos para conseguir un resultado verosímil sin sobrecargar más de lo necesario al computador.

- *Textura*: es una disposición de las partes de un cuerpo. En nuestro caso, en un documento existe una estructura bidimensional (representación plana), sin relieve.

- *Motor de renderizado*: programa que toma información de una escena digital para transformarla en una imagen 2D, útil para la creación de videos o realidad virtual. Traduce los parámetros de dicha escena tales como la incidencia de luz, formas geométricas, texturas, sombras, animaciones…

2.1.1 Funcionamiento

El renderizado es el proceso de conversión de un documento de tres dimensiones (*3D*) en una imagen de dos dimensiones (*2D*) que muestre un grado de realismo elevado. La imagen resultante se almacenaría en un archivo. El formato de ese archivo puede ser: PDF, *HyperText Markup Language* (*HTML*) que use *Javascript*, MOBI (derivado de la palabra inglesa *mobile*)... Entre esos archivos podemos hacer conversiones, por ejemplo, podemos convertir un archivo PDF a otro HTML o viceversa.

Nos centramos en archivos con formato PDF. Estos se pueden renderizar haciendo uso de distintas opciones de configuración. Algunas opciones de soporte del renderizador que ofrece Adobe Acrobat pueden ser:

- *Previsualizar mallas de renderizado personalizadas en la opción de menú vista.* Los modelados o ediciones de documentos en 3D se realizan mediante mallas de renderizado. Esta opción controla si las mallas de renderizado personalizadas producidas por determinados *plug-ins* se muestran en la vista.
- *Copiar configuración similar cuando se modifica el tipo de contenido*: determina si de manera predeterminada, la configuración del material, el entorno o la textura anterior se copiaría en el nuevo material, entorno o textura, cuando un tipo de contenido se cambie con el botón *Cambiar* del cuadro de diálogo *Editor de contenido*.
- *Comprobar si faltan texturas al renderizar*: si hay texturas referenciadas en el modelo que no están en el sistema, aparecería un cuadro de diálogo con una lista de las texturas que faltan.
- *Renderizados autoguardados*: los renderizados guardados se almacenan en una carpeta temporal. Se puede acceder a los archivos desde la ventana de renderizado, en el menú *Archivo > Recientes*.
- *Renderizados a mantener*: se trata del número de renderizados autoguardados que se mantendrían. Cuando se alcanza este número, los archivos anteriores se eliminan sin avisar.
- *Elegir el tamaño de textura* (*entre los valores:* 128, 256, 512, 1024 y 2048): cuando se muestra una textura algorítmica en la *vista*, se crea un mapa de bits (*bitmap*) [7] en el disco. Esta configuración determina el tamaño de la imagen en el disco. El uso de texturas más grandes puede mejorar la visualización de la *vista*; pero puede ralentizar determinadas operaciones.

2.1.2 Tipos de renderizado

Podemos distinguir dos modelos de renderizado o motores de renderizado muy utilizados para técnicas de diseño:

- *Unbiased*: el usuario no puede introducir ningún tipo de error, partiendo además de una solución muy ruidosa. Provocando que tengamos que depurar el resultado poco a poco hasta obtener el objetivo deseado.

- *Biased*: el usuario puede introducir de forma manual un nivel de error, en caso de ruido o soluciones borrosas para compensar la falta de información, especificando donde cree que con menos precisión de cálculo va a obtener los mejores resultados.

Mientras el primero es más preciso, sencillo de utilizar y tiene poco uso de *Random Access Memory* (*RAM*), el Biased es más rápido y versátil porque se puede configurar cualquiera de sus parámetros.

Para nuestro propósito pensamos que el método *Unbiased* es mejor ya que buscamos obtener el resultado óptimo depurando poco a poco cada documento, haciendo uso de muy poca memoria que puede ser destinada a otros procesos más costosos a lo largo de este TFG.

2.2 Reconocimiento óptico de caracteres

El OCR es una tecnología que le permite convertir diferentes tipos de documentos, tales como documentos en papel escaneados, archivos de PDF o imágenes captadas por una cámara digital en los datos editables y con opción de búsqueda. Es un sistema computerizado que permite escanear un documento de texto en un archivo digitalizado, que se puede editar con un procesador de textos del computador.

2.2.1 Funcionamiento

El paso previo del OCR es escanear para procesar un documento físico (en papel) entero (se procesan todas sus páginas), el OCR convierte el documento en una versión de dos colores (blanco y negro). La imagen escaneada (*bitmap*) se analiza en busca de áreas claras y oscuras, donde las áreas oscuras se identifican como caracteres que deben reconocerse y las áreas claras se identifican como fondo. Las áreas oscuras se procesan luego para encontrar letras alfabéticas o dígitos numéricos.

El software de OCR puede variar en sus técnicas, pero generalmente seleccionan: un carácter, palabra o bloque de texto. Los caracteres se identifican luego usando uno de estos dos algoritmos:

- *Reconocimiento de patrones* [8]: los programas de OCR reciben ejemplos de texto en varias fuentes y formatos que luego se utilizan para comparar y reconocer caracteres en el documento escaneado.

- *Detección de características*: los programas de OCR aplican reglas con respecto a las características de una letra o número específico, para reconocer los caracteres en el documento escaneado. Las características pueden incluir el número de líneas en ángulo, líneas cruzadas o curvas en un carácter para comparar. Por ejemplo, la letra mayúscula A puede almacenarse como dos líneas diagonales que se encuentran con una línea horizontal en el medio.

Cuando se identifica un carácter, se convierte en un código *American Standard Code for Information Interchange* (*ASCII*) que puede ser utilizado por los sistemas informáticos de manera más fácil. Los usuarios deben corregir los errores básicos y asegurarse de que los diseños complejos se manejan correctamente antes de guardar el documento para su futuro uso.

A continuación, presentamos los distintos pasos (metodología de trabajo) que sigue el OCR.

1. *Adquisición y obtención de una imagen binaria*: la adquisición se refiere al proceso de la toma de la imagen. Se puede realizar con una cámara fotográfica, con una de vídeo o con un escáner. Ha de obtenerse una representación binaria apta para la interpretación y procesado por medio de los aplicativos alojados en un *Personal Computer*, tableta gráfica... La obtención de una imagen binaria digital consiste en convertir la imagen digital en una imagen en blanco y negro, de tal manera que se preserven las propiedades esenciales de la imagen.

   La mayor parte de los algoritmos para reconocer escritura están escritos a partir de imágenes binarias, por lo que se hace conveniente el paso de una imagen en niveles de gris (o color) a una binaria, además esto permite reducir el volumen de los datos a tratar.

Uno de los métodos para poder obtener una imagen digital binaria es mediante el histograma de dicha imagen. A través del histograma obtenemos una gráfica donde se muestran el número de píxeles por cada nivel de gris que aparecen en esta. Para obtener la imagen digital binaria, se debe elegir un valor adecuado dentro de los niveles de grises (umbral), de tal forma que el histograma forme un valle en ese nivel. Todos los niveles de grises menores al umbral calculado se convertirán en negro y todos los mayores en blanco.

2. *Segmentación de la imagen*: operación que permite la descomposición de un texto en diferentes entidades lógicas. Estas entidades lógicas deben ser lo suficientemente invariables, para ser independientes del escritor, y lo suficientemente significativas para su reconocimiento.

   Una vez obtenida la imagen binaria se debe segmentar en las diferentes componentes conexas (parte de la imagen donde todos los píxeles son adyacentes entre sí) que la componen.

   La segmentación de la imagen constituye una de las mayores dificultades del reconocimiento, y se hace necesaria para poder reconocer cada uno de los caracteres de la imagen binaria.

   Con la segmentación se debería localizar las zonas de interés y separar las mismas. Las zonas de interés estarán caracterizadas por unos atributos comunes como son dimensión, superficie, densidad, inclinación, longitud del trazo...

   No existe ningún método genérico para realizar la segmentación de la imagen que sea lo suficientemente eficaz para el análisis de un texto. Los métodos mayormente utilizados son variaciones del método de *Proyecciones Vertical*. Otro ejemplo, sería utilizar un método que segmentase la imagen en entidades que no tuviesen ningún punto en común (*Tsujimoto*). Para imágenes de buena calidad el método produciría un resultado favorable. Sin embargo, en la realidad, teniendo en cuenta el ruido esto no se produce. Todos estos métodos no pueden ser aplicados al cien por cien, y se hace necesario el estudio del contexto para poder realizar una segmentación correcta. Pero el estudio del

contexto no se puede aplicar hasta que no hayan sido extraídos y reconocidos todas las entidades.

Una vez que se hayan extraídos todos los posibles cortes por algunos de los métodos de segmentación, se puede emplear un árbol de decisión. Para ello:

- Determinar todas las posiciones de cortes.
- Creamos un árbol de decisión, cada región entre dos cortes correspondería a un nodo del árbol.
- Asociamos a cada nodo un carácter y ponderaríamos su idoneidad respecto a esa región.

La segmentación de una imagen también se puede conseguir mediante la detección de fronteras o bordes. Algunos métodos son:

- El detector de bordes de Sobel.
- El operador de Kirsch.
- Detector de bordes de Marr-Hildreth.
- Detector de bordes de Canny.

Según el grado de asociación entre las operaciones de segmentación y las de reconocimiento, se distinguen tres tipos principales de métodos:

- *Explícitos o segmentación en unidades físicas:* intervienen avanzando el proceso de reconocimiento. Las partes segmentadas se dividen prácticamente en letras, tanto que la segmentación se considera una parte del proceso de reconocimiento.
- *Segmentación implícita o en unidades lógicas*: consisten generalmente en una segmentación más fina y así conseguir los puntos de corte correctos. Las partes segmentadas son llamadas grafemas. Estos se usan más adelante, durante el proceso de reconocimiento y clasificación. Los grafemas se componen de fragmentos de caracteres, caracteres o grupos de estos.

- *Segmentación implícitos y exhaustivos*: el reconocimiento guía a la segmentación, así que el sistema de evaluación que se aplica aquí implica un reconocimiento por cálculo de las posiciones sucesivas de la imagen y la elección de las posiciones de segmentación que se correspondan con las responsables de las partes más significativas.

3. Adelgazamiento de las componentes: obtenida las componentes conexas de la imagen se realiza un proceso de adelgazamiento de cada una de ellas. El procedimiento de adelgazamiento consiste en ir eliminando sucesivamente los puntos del borde de cada componente conexa, de forma que se preserve su topología. Las condiciones exactas que determinan si un punto se puede eliminar están relacionadas con el concepto de punto simple (aquel cuyo conjunto de los vecinos en negro tiene exactamente una componente conexa que es adyacente a él) y punto final (aquel que tiene exactamente un vecino negro y es un punto extremo del componente). Un punto del borde se elimina si es simple y no es final.

   La eliminación de puntos es un proceso iterativo en el que la componente mantiene la proporciones de la original. Este proceso se puede hacer en paralelo para cada componente, se muestra un ejemplo en la Figura 2.1.

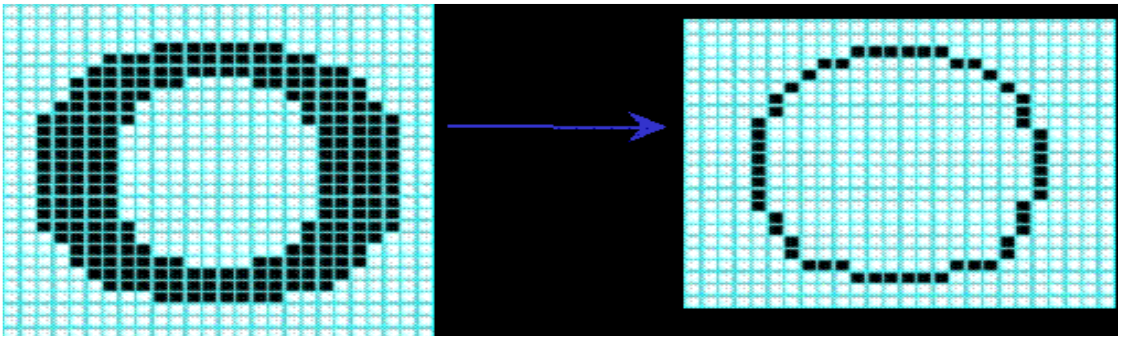


Figura 2.1. Proceso de adelgazamiento

4. Comparación con patrones: se compara los caracteres adelgazados con otros idealizados almacenados en una base de datos. Este paso es bastante importante, ya que el buen funcionamiento del *OCR* se debe en gran medida a una buena definición de esta etapa. Existen varios métodos para el reconocimiento de caracteres basados en la programación dinámica [9]:
    - *Geométricos o estadísticos*: extraer de los caracteres adelgazados un conjunto de *m* medidas que constituyen la composición de un vector de un espacio de representación (*Rm*) de dimensión *m*. Estas medidas suelen ser elevadas en número (*m>100*). Ejemplos de medidas son: topológicas y métricas, como superficies, regiones, perfiles, concavidades, bucles, intersecciones...
    - *Estructurales*: hacen el esqueletado de los caracteres extraídos de la imagen, de manera que pasan por una detección de contornos interiores y exteriores para detectar una serie de puntos singulares. De ellos se extrae información topológica. Posteriormente, se pasa a una vectorización que permite representar una descripción en forma de cadena de símbolos o de gráficos.
    - *Neuro-Miméticos*: basados en la utilización de redes neuronales de todo tipo: perceptrones multinivel, catas de *Kohonen*, *Time Delay Neural Network*, *Función de Base Radial*, neocognitrones, colonias corticales... La mecánica general es partir de una imagen con caracteres normalizados o de símbolos. Los resultados obtenidos son buenos con la condición de disponer de grandes bases de datos de aprendizaje. Como contrapartida, los tiempos de aprendizaje son elevados, además sería muy difícil determinar las causas de errores de estos métodos.

- *Markovianos*: se basan en hacer tablas de búsqueda de secuencias de señales de caracteres, pero variables en el transcurso del tiempo. Es algo parecido a una caché de reconocimiento de caracteres. Esto en cuanto a almacenamiento, pero el proceso de reconocimiento combina el uso de un grafo y un proceso aleatorio en las transiciones del grafo, así como en su retroalimentación (su recomposición a lo largo del tiempo de uso).
- *Basados en IA*: aplican sistemas de decisión a base de reglas al proceso de reconocimiento. Son algoritmos de búsqueda de árboles que sustentan la base de algoritmos clásicos (*A**).
- *Lógica difusa de Zadeh*: se basan en la lógica difusa, está muy bien adaptados al reconocimiento de caracteres con un tipo de datos impreciso. Usan máquinas de estados o reglas de Zadeh.
- *Mixtos*: combinación de algunos métodos anteriores, como por ejemplo un sistema *Compute Controllability Staircase Form*, que combina un árbol de decisión con una red neuronal.

2.2.2 Tipos de reconocimientos

En su significado más general el término OCR, se refiere a extraer texto de imágenes en todos los formatos posibles, ya sea una página impresa estándar de un libro o una imagen aleatoria con *graffiti* (en la naturaleza). En el medio, puede encontrar muchas otras tareas, como leer placas, captchas sin robot, letreros de calles... Cada uno de estos formatos tiene sus propias características que dificultan la obtención de su OCR; sin embargo, imágenes de paisajes tomadas en un ambiente real puede implicar un mayor grado de complejidad como se muestra en la Figura 2.2.

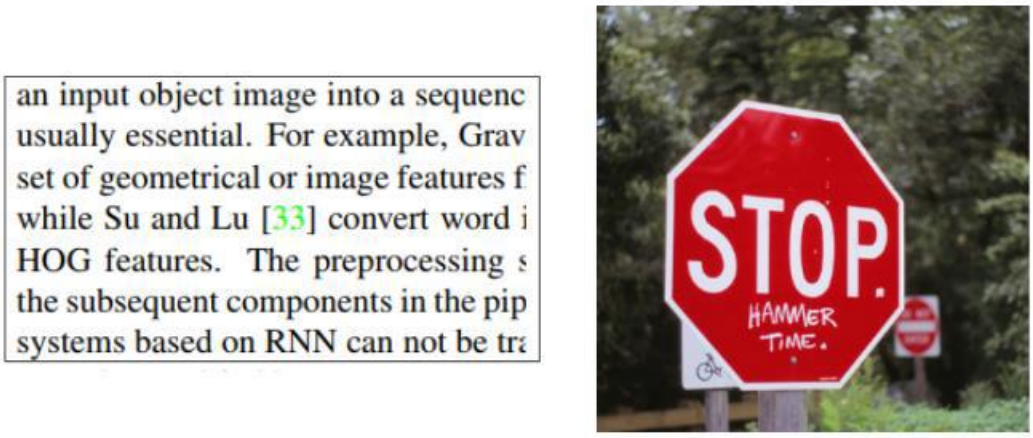


Figura 2.2. La naturaleza implica un mayor grado de dificultad

A partir de estos formatos, podemos diferenciar los distintos campos donde aplicar el OCR:

- *Densidad del texto*: en una página impresa o escrita, el texto es denso. Sin embargo, dada una imagen de una calle con un solo letrero, el texto es escaso.
- *Estructura del texto*: el texto en una página está estructurado, principalmente en filas estrictas, mientras que el texto en la naturaleza puede ser rociado en todas partes, en diferentes rotaciones.
- *Fuentes*: las fuentes impresas son más fáciles, ya que están más estructuradas que los textos escritos a mano.
- *Tipo de caracteres*: el texto puede estar escrito en un idioma diferente que puede ser muy diferente el uno del otro. Además, la estructura del texto puede ser distinta de los números, como los números de casa...
- *Artefactos*: evidentemente, las imágenes al aire libre son mucho más ruidosas.
- *Ubicación*: algunas tareas incluyen texto recortado o centrado, mientras que, en otras, el texto puede ubicarse en ubicaciones aleatorias en la imagen.

En función de la aplicación en dichos campos podemos distinguir distintos tipos de OCR:

1. *The Street View House Numbers*: un buen lugar para comenzar es *The Street View House Numbers* (*SVHN)* [10], que contiene el conjunto de datos de números de casas extraídos de *Google Street View*, tal y como indica su nombre.

   Esta tarea implica cierta dificultad. Por lo general, los dígitos de cada casa siempre se representan en varias formas y estilos de escritura, sin embargo, los números de las casas se suelen siempre encontrar en la misma posición, por tanto, en casos muy similares se eliminaría el uso de la detección. Las imágenes no tienen una resolución muy alta y su disposición puede ser un poco peculiar (Figura 2.3).

2. *Matrículas*: requiere en primer lugar realizar la detección, en este caso la de la placa del vehículo y luego reconocer sus caracteres. Dado que la forma de la placa es relativamente constante, algunos enfoques utilizan un método de remodelación simple, antes de reconocer realmente los dígitos. Aquí hay algunos ejemplos de la web además de lo como se muestra en la Figura 2.4:

   - *OpenALPR* [11] es una herramienta muy robusta, sin un aprendizaje profundo involucrado, para reconocer las placas de varios países.
   - Este [12] proporciona una implementación del modelo de *Red Neuronal Recurrente Convolucional*.
   - *Supervise.ly*: compañía de servicios de datos escribió sobre la capacitación de un reconocedor de matrículas utilizando datos artificiales generados por su herramienta (los datos artificiales también se discutirán más a fondo) [13].
   - *VPAR SERVER* de Neural Labs. [14]: una biblioteca de reconocimiento de matrículas VPAR que trabaja de forma autónoma, resuelve internamente la conectividad con las cámaras y guarda los resultados en su propia base de datos MS

SQL. Pensado para empresas que requieren incorporar reconocimiento de matrículas, de manera rápida y segura, sea cual sea el escenario, ya sea embarcando en vehículos policiales, cámaras en puntos fijos, controles de acceso, parking…

Reconoce de forma continuada matrículas de vehículos tanto en movimiento *(Free-Flow)* como parados *(Stop & Go),* en la Figura 2.5 lo podemos mostrar**.** Permite conectar cámaras IP de diferentes fabricantes y protocolos al mismo sistema.

Figura 2.3. Ejemplo de SVNH

Figura 2.4. OCR aplicado a matrículas de vehículos

Figura 2.5. Uso del OCR en Parkings

3. *OCR en la naturaleza*: una de las tareas de *OCR* más desafiante, ya que introduce todos los desafíos generales de visión por computadora, tales como el ruido, la iluminación y los artefactos en *OCR* como se ve en la Figura 2.6. Algunos de estos conjuntos de datos relevantes para esta tarea son el *coco-text* [15], y el conjunto de datos *The Street View Text Dataset* (*SVT*) [16] que, una vez más, utiliza imágenes de *Street View* para extraer texto.
4. *Texto del sintetizador*: *SynthText* [17] crea palabras al azar y las introduce en una imagen. Debido a que generar texto en la naturaleza es complejo, *SynthText* tiene en cuenta la profundidad de la imagen introduciendo anotaciones en forma de palabras alrededor de la imagen de forma inteligente.

   En la Figura 2.7 se muestra una imagen del proceso de SynthText: arriba a la derecha apreciamos la segmentación de una imagen, abajo a la derecha están los datos de profundidad. En la parte inferior izquierda hay un análisis de la superficie de la imagen, que según el texto se rocía en la imagen.

Figura 2.6. Ejemplo OCR en la naturaleza

Figura 2.7. Proceso SynthText

La biblioteca *SynthText* toma con cada imagen dos máscaras, una de profundidad y otra de segmentación. Si se desea utilizar sus propias imágenes, también se deben agregar estos datos.

5. *PDF OCR: normalmente el OCR se aplica a documentos impresos o archivos en formato PDF. La mayoría de las herramientas de OCR (por ejemplo, Tesseract [19]) están destinadas principalmente a que podamos aplicar el OCR a un archivo en formato PDF.*

   Por último, destacar que la base de datos *MINST* [18] es un conjunto de dígitos escritos a mano como podemos mostrar en la Figura 2.8 que ha sido usado durante años para la investigación en *ML*. Contiene un conjunto de entrenamiento de 60.000 muestras y un conjunto de pruebas de 10.000 muestras. La ventaja de usar un conjunto de datos como *MINST* para hacer experimentos es que viene de serie con la mayoría de las herramientas de *ML*.

Hemos explicado distintas opciones de *OCR* y la base de datos *MINST* para que se comprenda mejor, en general, la problemática del *OCR*. Nos centramos en el *PDF OCR* con la peculiaridad que la mayoría de *PDF*, por no mencionar que la totalidad de ellos, con los que trabajamos son documentos compuestos de imágenes escaneadas.

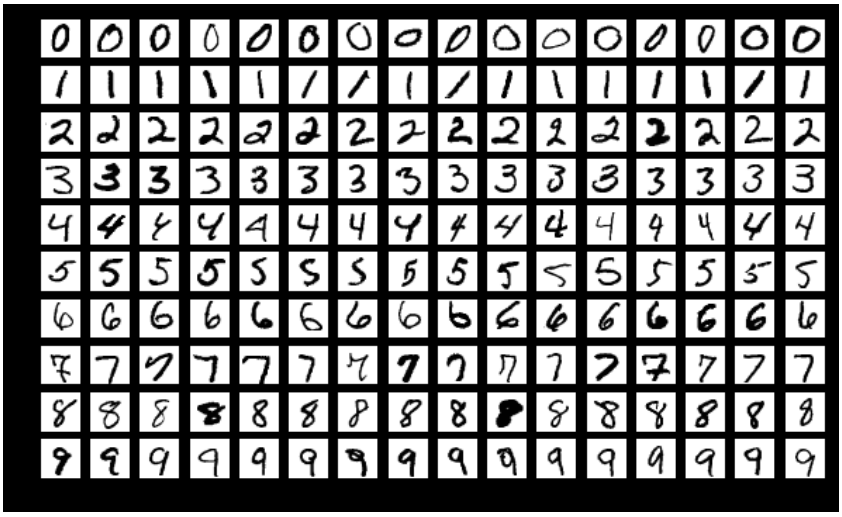


Figura 2.8. Base de datos MINST

2.3 Técnicas de Machine Learning

El ML es una disciplina del campo de la IA en el que con el uso de algoritmos permite a los ordenadores ser capaces de identificar patrones en grandes bancos de datos y elaborar a través de las mismas predicciones.

2.3.1 Funcionamiento

Una técnica de IA es el *Aprendizaje Automático* (*AA*). Ejemplos de AA son: a) los algoritmos genéticos para, por ejemplo, la planificación de procesos y hebras que realizan los sistemas operativos, b) las redes neuronales, para modelizar y comprender el aprendizaje humano, pero también para tareas básicas de control y supervisión básicos en la Industria y c) los métodos probabilísticos y bayesianos, por ejemplo, para llevar a cabo la búsqueda en grandes bases de datos.

Esta capacidad de descubrimiento de la IA se ha ampliado debido al incremento masivo de los datos digitales, la capacidad de computación y las innovaciones en los algoritmos de IA como se muestra en la Figura 2.9.

Figura 2.9. Ejemplo de Uso de IA

OCR	ICR
El OCR se basa en plantillas para la extracción de los datos.	Utiliza redes neuronales de inteligencia artificial para extraer datos.
Utiliza un formato específico para la entrada de datos.	Reconoce varios formatos.
Ideal para empresas que tienen una estructura fija para documentos.	Se adapta y está capacitado para cambios frecuentes.
La intervención manual y la revisión son necesarias para los sistemas basados en OCR.	Solo marca anomalías y pide a los usuarios que verifiquen cuando sea necesario.

Tabla 2.1. Diferencias entre OCR e ICR

El *Intelligent Character Recognition* (*ICR*) es una tecnología que se utiliza para extraer texto escrito a mano de archivos compuestos de imágenes. Es una versión avanzada de la tecnología OCR en la que los algoritmos de AA y la IA interpretan de manera inteligente los datos en formularios y documentos físicos mediante el reconocimiento de documentos escritos a mano con distintos estilos. La tecnología ICR se utiliza para escanear y extraer información de documentos en papel y almacenar los datos digitalmente obtenidos en una base de datos. Los datos se utilizan para informes analíticos donde se describe cómo se pueden organizar esos datos y obtener información en tiempo real a partir de estos. ICR al igual que el OCR permite a los usuarios leer rápidamente información escrita a mano en papel y convertirla a formato digital entre otras diferencias que podemos observar en la Tabla 2.1. Los motores ICR funcionan con OCR para automatizar la captura de datos eliminando la necesidad de presionar teclas para ingresar datos. Cuenta con un alto nivel de precisión.

La automatización de procesos es una tecnología que se utiliza para configurar el software ICR y ejecuta tareas sin errores. El reconocimiento de patrones, la IA y el reconocimiento de voz son características implementadas con la automatización de procesos robóticos e ICR para ampliar sus capacidades. Los informes generados son precisos y el ICR permite a los humanos obtener información a partir de análisis sofisticados que ayudan a tomar decisiones claves.

Si bien puede parecer que ICR reúne todas las características para que llevemos a cabo su implementación y elección para este proyecto, la falta de documentación e información sobre el mismo y que podamos distinguir los pasos del OCR y del ML independientemente el uno del otro, nos han llevado a descartar ICR para este TFG.

2.3.2 Tipos de Machine Learning

El desarrollo de aprendizaje automático se centra en tres categorías principales que se conocen como aprendizaje supervisado, aprendizaje no supervisado, aprendizaje profundo.

- *Supervisado:* el proceso de generación de conocimiento se realiza con un grupo de ejemplos o datos etiquetados, en los que los resultados que arroja la operación son conocidos previamente. Este tipo de modelo aprende de estos resultados e incorpora ajustes en los parámetros interiores para poder adaptarse a datos nuevos que ingresan al sistema.

  Gracias al aprendizaje desarrollado por estos modelos supervisados se alimenta un conjunto de resultados, que permite realizar predicciones adecuadas del comportamiento de datos nuevos que aún no han sido procesados. Este tipo de aprendizaje es el que se incorpora en aplicaciones tecnológicas como filtros detectores de *Spiced Ham* (*SPAM*) en correos electrónicos, detectores de imágenes en captchas o en aplicaciones de reconocimiento de voz o escritura.
- *No Supervisado:* incluye conjuntos de datos sin etiquetar en los que no se conoce previamente la estructura que estos poseen. En este tipo de aprendizaje se busca obtener información clave o importante, sin conocer previamente la referencia de las variables de salida, explorando la estructura de los datos que no están etiquetados.

Categorías:

- *Clustering*: técnica exploratoria para analizar datos en la que se organiza la información por grupos, desconociendo de forma previa la estructura que los compone. Esto se hace con la finalidad de obtener grupos de datos con características similares. Es usual utilizar este tipo de análisis de datos en estrategias de *marketing* ya que facilitan la construcción de segmentos o nichos de mercado utilizando variables específicas para su análisis.

- *Reducción dimensional*: se utiliza con datos de alta complejidad que demandan mayor capacidad de procesamiento. Esta funciona determinando correlaciones entre las características que se presentan en los conjuntos de datos, disminuyendo las redundancias de información y reduciendo el tiempo de análisis para poder obtener de forma más eficiente la información considerada de mayor valor.

- *Reforzado:* construye modelos que aumentan el rendimiento tomando como base el resultado o la recompensa que se genera por cada interacción realizada. Esta recompensa es el producto de una acción correcta o conjunto de datos devueltos que entran en una medida especifica. El modelo mediante un agente utiliza la recompensa como parámetro de ajuste en su comportamiento para acciones futuras, de tal forma que la acción nueva cumpla igualmente con el objetivo o acción correcta y así obtener una recompensa máxima. Forma parte del *Deep Learning*.

Para nuestro propósito, el aprendizaje supervisado es mejor ya que este tipo de modelo aprende de los resultados e incorpora ajustes en los parámetros anteriores para poder adaptarse a datos nuevos que ingresan al sistema realizando así un entrenamiento y validación de los datos a tratar en este TFG.

3. Herramientas utilizadas

En este capítulo se describen las características más importantes de las tecnologías utilizadas para resolver el problema, haciendo hincapié en aquellos aspectos más destacados que justifican su uso y posterior aplicación para la solución aportada.

3.1 Python

Los paradigmas de la programación son:

- *Imperativo*: describen el estado del programa y permiten su modificación mediante condiciones o instrucciones de código que le indican al computador cómo realizar una tarea. Se describe paso a paso un conjunto de instrucciones que deben ejecutarse, para variar el estado del programa y solucionar el problema.

- *Funcional*: basada en el uso de funciones matemáticas que permite la variación del programa mediante la mutación de variables.

- *Declarativo*: declaran condiciones, ecuaciones… que describen un problema y detallan su solución.

- *Orientado a objetos*: manipulan los objetos de entrada para la obtención de resultados (salida) específicos donde cada objeto nos ofrece una función específica y también nos permite la agrupación de bibliotecas o bibliotecas. Los objetos son entidades que tienen un determinado estado, las entidades son propiedades que los diferencian.

Los lenguajes interpretados son aquellos en los que el código del programador es traducido mediante un intérprete a medida que se va ejecutando. Un lenguaje con tipado dinámico es aquel que permite que una variable pueda tomar diferentes valores de distintos tipos en diferentes momentos. Esto permite cambiar el valor y tipo de una variable durante la ejecución sin necesidad de volver a declararla.

Python es un lenguaje imperativo, funcional y orientado a objetos interpretado que tiene tipado dinámico (las variables son declaradas por su contenido y no por su contenedor), es multiplataforma y multipropósito. Por ser interpretado presenta ciertas ventajas:

- Al ser interpretado no es necesario compilar ahorrando mucho tiempo en el desarrollo y prueba de una aplicación.

- El código fuente se puede ejecutar en cualquier entorno software siempre y cuando este disponga del intérprete (Windows, Linux, Mac, Android, Web...).

Dispone de numerosas bibliotecas que facilitan la programación de diversos tipos de aplicaciones.

Como la mayoría de los scripts o códigos que hemos encontrado implementado en Python, o se puede desarrollar en él, decidimos realizar este proyecto en este lenguaje. Facilitando mucho así la implementación del renderizado de los PDF en dicho lenguaje, así como el uso de OCR y obtención de palabras claves que en la mayoría de los casos se realizan en este lenguaje de programación.

3.2 Poppler

Las recomendaciones técnicas que hacen las distintas instituciones implicadas en procesos de digitalización en relación con el OCR son muy importantes. Estas recomendaciones se refieren principalmente a la resolución mínima de la imagen escaneada porque es un factor determinante para obtener un resultado satisfactorio: a mayor resolución de escaneo, mayor precisión del OCR. Con carácter general, se establece una resolución mínima de 300 ppp para que el reconocimiento de los caracteres sea efectivo como se muestra en la Figura 3.1, aunque dependiendo de las características del documento se aconseja una resolución mínima superior.

Tipo de documento	Resolución mínima
Textos con tipos de letra claros	300 ppp
Tipos de letra pequeña u originales de poca calidad (prensa)	600 ppp

Figura 3.1. Recomendaciones técnicas para la aplicación del OCR

Poppler [20] es una biblioteca (implementada en Python), que renderiza los PDF basado en *xpdf-3.0* [21]. Xdpf-3.0 es un visor de PDF y un kit de herramientas gratuitos, que incluye un extractor de texto, un conversor de imágenes, un conversor de HTML... La mayoría de ellas están disponibles como código abierto.

Utilizamos Poppler [22] a través de la ventana de órdenes dando buenos resultados. Para integrar su uso en Python utilizamos su biblioteca *pdf2image*, su implementación se muestra en la Figura 3.2: un módulo basado en *Poppler* que integra a *pdftoppm* y *pdftocairo* para convertir PDF a un objeto de imagen *Python Imaging Library* (*PIL*).

PIL es una biblioteca de Python gratuita que permite la edición de imágenes. Soporta formatos tales como *Graphics Interchange Format* (*GIF*), *Joint Photographic Experts Group* (*JPEG*) y *Portable Network Graphics* (*PNG*). Esta biblioteca solo soporta hasta la versión 2.7 de Python, aunque exista una versión que mantiene la biblioteca y hace que se pueda en versiones de Python superiores como el nuestro (Python 3.x).

Hay que destacar que para todos los sistemas operativos es necesario tener instalado previamente Poppler.

Por tanto, elegimos Poppler por los resultados que proporciona, la facilidad para lograr su implementación, así como por ventana de órdenes, su disponibilidad en todos los sistemas operativos y ser de código abierto.

```
from pdf2image import convert_from_path, convert_from_bytes
from pdf2image.exceptions import(
    PDFInfoNotInstalledError,
    PDFPageCountError,
    PDFSyntaxError
)
```

Figura 3.2. Import de la biblioteca de Poppler en Python

3.3 ImageMagick

Ahora abordamos más aspectos como el contraste y la iluminación para mejorar la imagen de entrada al OCR. Dependiendo de cada imagen buscamos tratar la imagen de una manera u otra buscando en algunas por ejemplo obtener una imagen con más brillo y más contraste.

Consideramos la herramienta *ImageMagick* [23] [24] (la guía de instalación se puede consultar en [25]) que es un conjunto de utilidades de código abierto para mostrar, manipular y convertir imágenes, bajo licencia de Apache y disponible para todos los sistemas operativos, además de que se puede ejecutar por ventana de órdenes con gran facilidad. Entre sus características podemos destacar:

- *Multiplataforma en modo consola:* se puede hacer scripts que automaticen los procesos, bibliotecas o funciones que permiten repetir automáticamente los efectos sobre varias imágenes.
- No necesita un entorno gráfico.
- *Multiformato*: trata más de 200 formatos gráficos: (mapa de bits, PNG, *Joint Photographic Group* (*JPG*), *eXperimental Computing Facility* (*XCF*), *Power Spectral Density* (*PSD*), *Program Specification Block* (*PSB*), *en inglés "crudo"* (*RAW*), *Tagged Image File Format* (*TIFF*), *Scalable Vector Graphics* (*SVG*), *Magick Vector Graphics* (*MVG*)...) y de vídeo (*QuickTime Movie* (*MOV*), *Moving Picture Experts Group* (*MPG*), *MPEG-4* (*MP4*) ...), así como PDF….
- *Más de 20 espacios de color distintos*: *Red Green Blue* (*RGB*), *Red Green Blue Alpha* (*RGBA*), *Cyan Magenta Yellow y Key* (*CMYK*), *Hue Saturation Value – Matiz Saturación Valor* (*HSB*), *Hue Saturation Lightness* (*HSL*)... ofreciendo muchas más posibilidades que la mayoría de los programas de edición de imágenes. Permite trabajar individualmente con cada uno de los canales de cada espacio de color, tanto extrayendo el canal en cuestión como aplicando las modificaciones que deseemos únicamente a un canal, consiguiendo resultados mucho más precisos que trabajando con la imagen completa.
- Permite cambiar de espacio de color (y, por lo tanto, trabajar con los canales de ese espacio de color) sobre la misma imagen en la misma instrucción. Es decir,

podemos aplicar, por ejemplo, modificaciones en la capa de luminosidades del espacio *Lab*, otras independientes al canal rojo en el espacio RGB, otros cambios distintos a la capa de saturación en un espacio *HSB* y que el resultado de esos cambios lo transforme a un espacio de color de escala de grises.

Para poder implementar el ImageMagick en Python y editar las imágenes apropiadamente, buscando eliminar defectos innecesarios como ruido, marcas de agua... Utilizamos la biblioteca Wand que es un *ctypes* (biblioteca externa de phyton) [26] para Python basado en ImageMagick. Comentar a su vez que el uso de esta biblioteca implica que también se utilicen módulos matemáticos como *math* [27] en la implementación del código como se puede ver en la figura 3.3.

Hay que destacar que esta fase se puede realizar de diferentes maneras: hacer uso de *Poppler* para editar la imagen obtenida a partir de la renderización del PDF mediante la herramienta GIMP [28], la cual permite manipular imágenes con múltiples combinaciones. Sin embargo, para que nuestro trabajo no sea muy tedioso en cuanto a editar imagen a imagen, en vez de tratarla como bloques de imagen, preferimos utilizar *ImageMagick* que es más eficiente.

```
from wand.image import Image
class MyImage(Image):
    def brightness_contrast(self, brightness=0.0, contrast=0.0):
        slope=math.tan((math.pi * (contrast/100.0 +1.0)/4.0))
        if slope < 0.0:
            slope=0.0
        intercept=brightness/100.0+((100-brightness)/200.0)*(1.0-slope)
        self.function("polynomial",[slope,intercept])
```

Figura 3.3. Import de Image Magick en Python

3.4 Tesseract

Entre las herramientas más destacadas para aplicar el OCR están:

1. *Nuance OmniPage* [29]: convierte de forma rápida y precisa documentos en papel, archivos PDF e incluso fotos digitales en archivos editables. Es compatible con casi cualquier escáner, e incluso permite capturar documentos con una cámara digital, en iPhone o iPad.
2. *ReadIRIS Pro* [30]: de gran alcance diseñada para usuarios domésticos y profesionales. Al igual que *ABBYY* y *Nuance OnmiPage* también es de pago.
3. *ABBYY Fine Reader OCR* [31]: permite reconocer todos los caracteres de una imagen o un documento PDF, extraerlos y permitirnos copiarlos y trabajar con ellos como si fueran texto plano. Esta es una de las herramientas más efectivas, con una tasa de acierto muy elevada, y compatible con más de 190 lenguajes diferentes. Además, se integra perfectamente con Microsoft Word de manera que, si escaneamos un documento, automáticamente podamos tenerlo en forma de texto en la herramienta de Microsoft.

   Aunque este es, probablemente, el programa más eficaz en este aspecto, el principal problema es que es de pago, y no precisamente barato (200 euros la versión más limitada en funciones), por lo que si estamos buscando un programa gratuito que nos permite convertir nuestros escaneos a texto, podemos probar cualquiera de las siguientes alternativas gratuitas.
4. *GImageReader* [32]: frontend que utiliza la biblioteca mencionada anteriormente y que permite hacer uso de las funciones de reconocimiento de una forma muy sencilla e intuitiva.
5. *(a9t9) Free OCR Software*: de código abierto, puede ejecutarse directamente desde el navegador sin necesidad de instalar ningún software adicional.
6. *Free OCR to Word* [33]: permite reconocer los caracteres de distintos formatos de archivos, como *JPG*, *JPEG*, *PSD*, *PNG*, *GIF*, *TIFF* y *Bit Mapped Picture* (*BMP*), entre otros, e importarlos directamente a un documento de Word totalmente editable de manera que evitemos la tediosa tarea de reescribir estos documentos.

7. *Ocropy* [34] (denominado *OCRopus*): sistema de OCR escrito en *Python*, *NumPy* y *SciPy* que se centra en el uso del AA a gran escala para abordar problemas en el análisis de documentos. Se puede usar desde la línea de órdenes o dentro de *gscan2pdf*. Tiene como principal función la de trabajar gran volumen de datos, como por ejemplo para la búsqueda de libros de Google, pero también se suele usar para personas con discapacidad visual. La base de código está principalmente en C ++, con algo de *Python*.
8. *Cuneiform* [35]: sistema de reconocimiento de caracteres ópticos de código abierto y en varios idiomas desarrollado originalmente por Cognitive Technologies. Este paquete de software también realiza análisis de diseño y reconocimiento de formato de texto. Su versión para Linux no tiene un componente de interfaz gráfica, pero se han desarrollado interfaces gráficas de usuario. Conserva la estructura del documento y su formato. El programa reconoce la tabla de cualquier estructura y complejidad, incluso sin mostrar las líneas de la cuadrícula. Los resultados del programa se pueden editar en aplicaciones de Office, editores de texto y guardar en formatos populares para realizar búsquedas de texto completo.
9. *Ocrad* [36]: basado en un método de extracción de caracteres. Lee imágenes en formatos *BMP*, *PGM* (escala de grises) o *ppm* (color) y produce texto en formato byte (8 bits) o 8-bit *Unicode Transformation Format* (*UTF-8*). Incluye un analizador de diseño capaz de separar las columnas o bloques de texto que normalmente se encuentran en las páginas impresas. Se puede usar como una aplicación de consola independiente o como un backend para otro software. Su uso es principalmente para fines de investigación. Reconoce a los personajes por su forma, y la razón por la que es tan rápido, es que no compara la forma de cada personaje con algún tipo de base de datos de formas y luego elige la mejor coincidencia. Solo compara las diferencias de forma que son relevantes para elegir entre dos categorías de caracteres, principalmente como una búsqueda binaria.
10. *GOCR* [37] (bajo la Licencia Pública General de *GNU*): lee imágenes en muchos formatos y genera un archivo de texto. Proporciona una interfaz

gráfica simple escrita en *Tcl* / *Tk* y algunos archivos de muestra. Reconoce y traduce códigos de barras.

11. *Tesseract* [38] (biblioteca gratuita y *OpenSource*): motor de *OCR*, cuenta con repositorio en *GitHub y* tiene una licencia de software libre Apache. Compatible con cualquier sistema operativo, tiene un desarrollo continuo.

Existen productos de ML que ofrecen herramientas OCR pero que no analizamos porque se escapa de nuestro interés.

Debido a que no vamos a seleccionar ningún sistema que se encuentre en la Nube, descartamos dichos sistemas (entre otros Free OCR Software). Además, descartamos los que sean de pago (Nuance OmniPage, ReadIRIS Pro y ABBYY Fine Reader OCR). Eligiendo, por tanto, en primer lugar, los que se ejecuten en el computador local. Los que cumplen esta propiedad son: GOCR, GNU Ocrad Y Tesseract. En la Tabla 3.1 se muestran las ventajas e inconvenientes de estas herramientas. Además, en [39] se muestra sus valoraciones.

Elegimos Tesseract que es una biblioteca, que ofrece numerosos ejemplos e implementaciones en Python y en Java, además de las ventajas y desventajas que tiene en comparación con las otras herramientas como vimos en la tabla anterior, además en el Anexo B se presentan las diferencias y ventajas que ofrece Tesseract con los sistemas descartas. Aunque podría resultar un poco incómoda de utilizar ya que su uso debe hacerse desde terminal o desde una ventana de órdenes, (aunque también cuenta con bibliotecas en Pyhton); sin embargo, las órdenes son muy sencillas y el resultado que nos ofrece es excelente a nivel de precisión.

	Tesseract	Ocrad OCR, GOCR.
Ventajas	Licencia Apache Gratis Disponible en Windows,Linux y OSX Detector de 40 Idiomas FrontEnds disponibles	Implementación muy simple Múltiples interfaces Compatible con Windows, Linux y OSX
Desventajas	-	Guarda los archivos en un formato nativo No funciona bien con diseños multilinea

Tabla 3.1. Tabla ventajas e inconvenientes Tesseract y Ocrad.

3.4 WordCloud y NLTK

Una nube de etiquetas (*tags*) es la representación visual de un grupo o familia de tags agrupadas en un conjunto abstracto que se asemeja a una nube y que tienen un contexto o punto de unión común. Su objetivo principal de la nube de tags, el de facilitar al usuario la búsqueda de información, al indicarle de manera recurrente las palabras y áreas más destacadas, actuales o abundantes en contenido relevante para él, representadas por esas etiquetas. El grupo de tags se dispone en diferentes ubicaciones, formas, tamaños o colores dentro de la nube. Normalmente las de mayor tamaño y colores intensos, reflejan las temáticas de mayor importancia (por relevancia, volumen de contenido o actualidad) siendo las menos significativas aquellas más pequeñas y de colores más degradados (Figura 3.4).

Existen numerosas páginas y aplicaciones web que permiten ayudar a crear estas nubes de tags, algunas son herramientas online que permiten crear las nubes, modificarlas cambiar su estilo de diseño, formas, fuentes, tamaños o colores. Algunas de ellas pueden ser *Wordle* [40], *Wordaizer* [41], *Tagxedo* [42] o *PytagCloud* [43] entre muchas otras. De estas, descartamos *Tagxedo* por lo mismo que hicimos con las herramientas de OCR, ya que realiza el proceso online y no nos interesa por la privacidad de los datos a utilizar:

Figura 3.4. Ejemplo de Nube de Tags

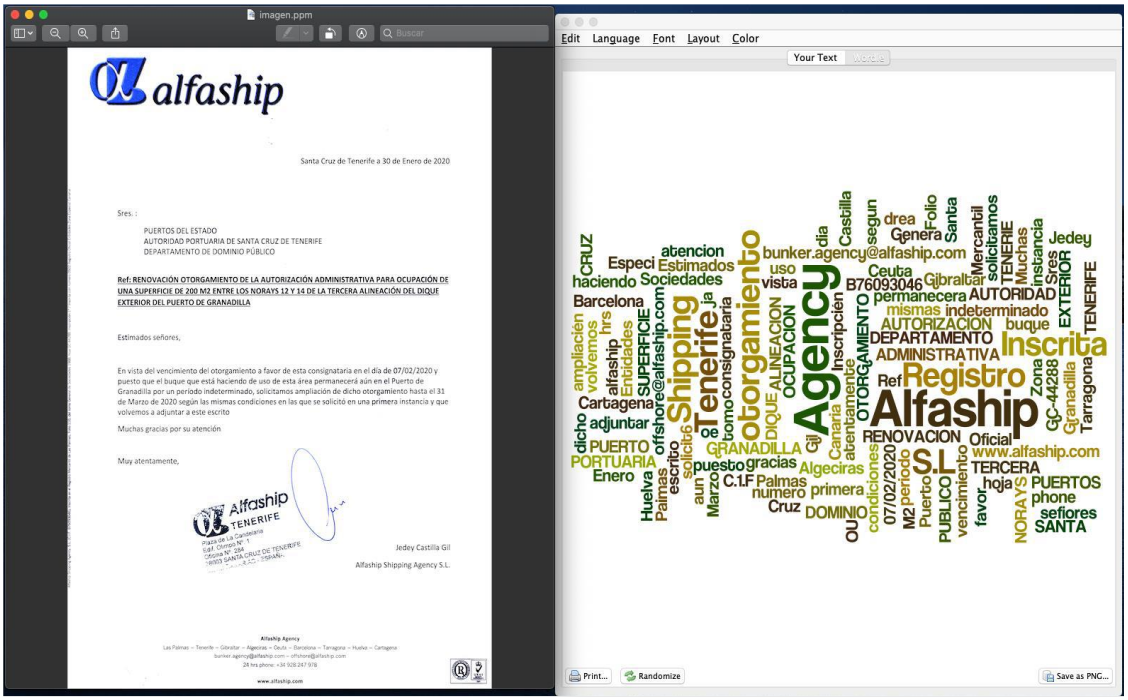


Figura 3.5. Uno de nuestros documentos al que se le aplica el OCR y la Nube de Tags

1. *Wordle* [*44*]: genera nubes a las que se les puede dar diversos formatos visuales, a partir de un texto cualquiera elegido por el usuario. No comparte su código (el código pertenece a IBM).

   En la Figura 3.5 se muestran las palabras más significativas obtenidas tras realizar el OCR a la imagen de la izquierda, se representan de mayor a menor tamaño indicando cuál es su importancia.

   Debido a su disponibilidad en diferentes plataformas elegimos Wordle.

2. *PyTagCloud*: permite crear una nube de tags simple basada en Wordle. Las salidas que nos presenta PyTagCloud pueden ser tanto en PNG como en HTML/CSS.

   Su instalación se realiza haciendo uso de *pip* [*45*] en nuestro equipo. Como queremos configurar la nube de tags a nuestra voluntad para así poder configurar también el diccionario con las palabras o tags que más nos interesan, basamos la creación de dicha nube en el uso de unas bibliotecas especializadas en *Procesamiento del Lenguaje Natural* (*PLN*) y en el análisis de textos. La biblioteca que empleamos para las pruebas es *Natural Language Toolkit* (*NLTK*) [*46*] [*47*].

```
#Importación de las librerías NLTK y Gensim
import nltk
import gensim
```

Figura 3.6. Import de la biblioteca nltk

Además, usamos la biblioteca *Gensim* [*48*] que empleamos para el análisis textual cuya importación en nuestro código se muestra en la Figura 3.6. Para la correcta instalación de la biblioteca NLTK es necesario ejecutar la orden de la Figura 3.7.

Al ejecutarla, se abre una ventana emergente con los complementos de NLTK, como se muestra en la Figura 3.8. En ella se pueden seleccionar los más deseados. En nuestro caso los seleccionamos todos marcando la casilla "ALL".

Se observa que no lo tenemos instalado así que realizamos su instalación, la cual conlleva unos minutos como se observa en la interfaz que nos ofrece la herramienta (Figura 3.9).

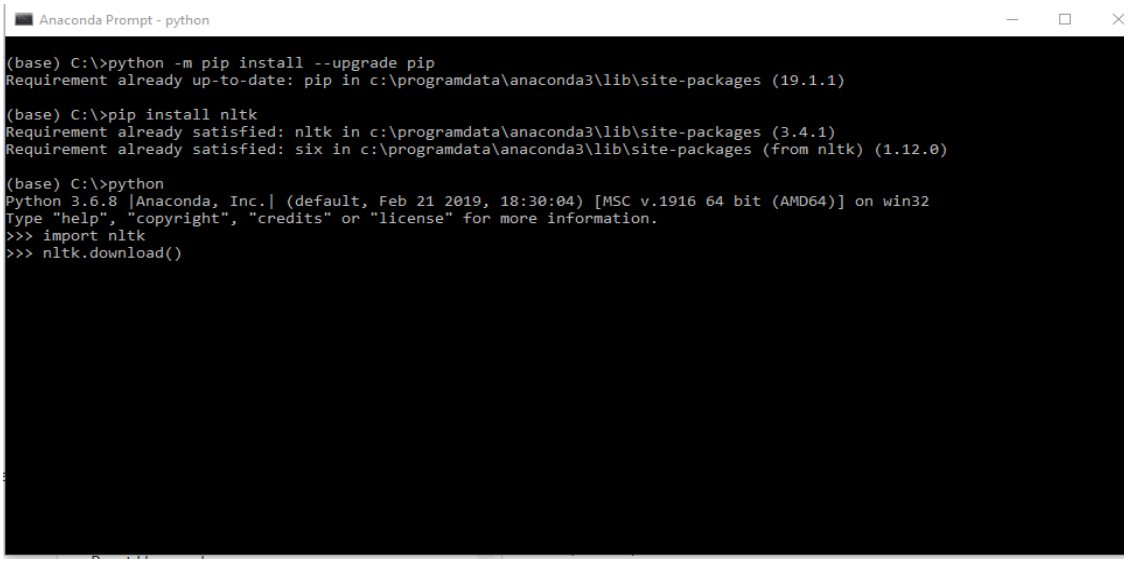


Figura 3.7. Instalación del NLTK

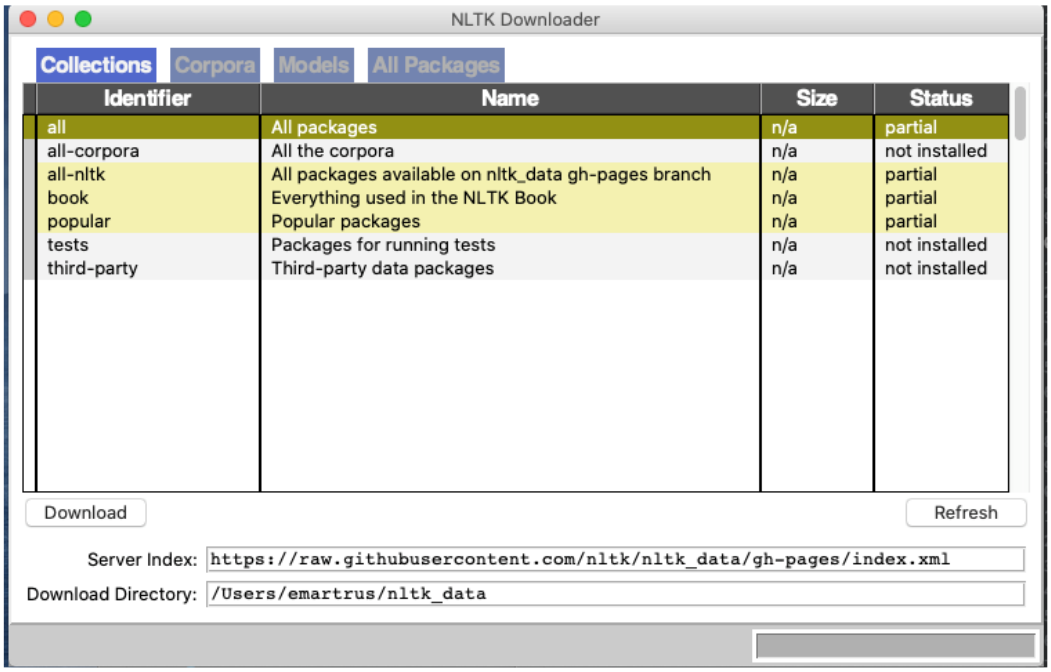


Figura 3.8. Instalador de NLTK

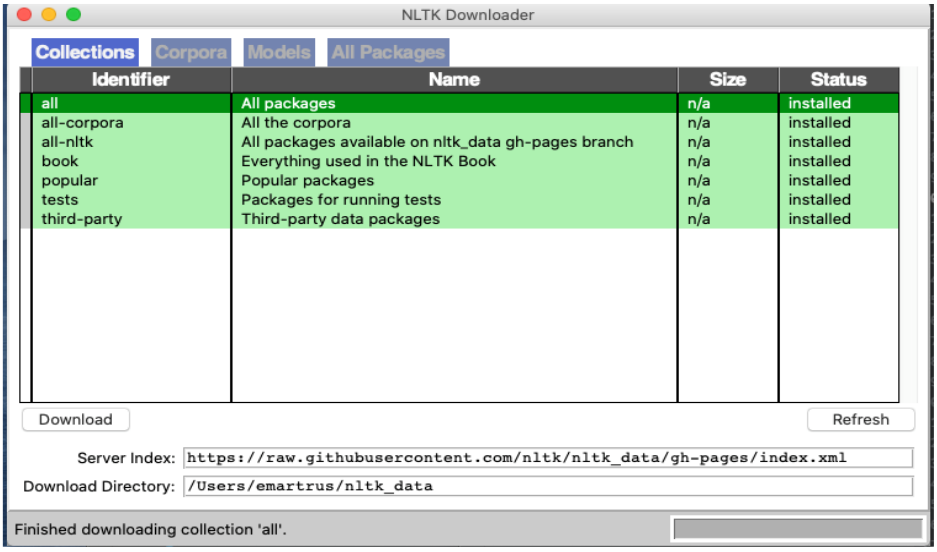


Figura 3.9. Proceso de instalación de NLTK

En nuestro caso obtenemos errores con los certificados SSL, indicándonos que los debemos instalar mediante la orden *Install Certificates.command*. Por ejemplo, basta con ejecutar una orden como:

- */Applications/Python\ 3.6/Install\ Certificates.command*

Solucionando los problemas que se nos presentan al ejecutar el script.

El uso de NLTK nos permite descartar todas aquellas palabras de uso cotidiano y quedarnos con las que más no interesan, siendo muy simple de implementar y con la posibilidad de cambiar al idioma que deseemos en cualquier momento.

3.5 Weka

Entre las principales herramientas de ML que podemos encontrar para realizar la clasificación de las palabras más determinantes en los departamentos correspondientes en nuestro TFG podemos analizar las siguientes:

1. *Azure Machine Learning Studio* [*49*]: cuenta con una interfaz de arrastrar y soltar que no requiere experiencia en codificación. Pero se necesita un enfoque aplicado al aprendizaje automático.

   Permite integrar la tecnología en su trabajo rápidamente. La interfaz visual también permite exportar datos relacionados con el análisis predictivo. Por lo que es sencillo compartir sus hallazgos con los miembros de la junta ejecutiva u otros superiores.

   Hay una versión gratuita disponible que le permite experimentar con el programa y cómo funciona. El precio comienza en $ 9.99 por mes [*50*].
2. *Amazon Machine Learning* [*51*]: presenta la misma tecnología utilizada internamente por sus científicos de datos. En este momento se encuentra disponible para los clientes que se suscriben al servicio. Es una opción ideal para los científicos de datos que necesitan confiar en el aprendizaje automático para cumplir con plazos ajustados.

   Ofrece tres capacidades de predicción de aprendizaje automático. Con lo cual no es necesario conocer ningún método de aprendizaje automático

antes de importar datos. La herramienta analiza la información y elige la mejor para usted.

Sin embargo, la fijación de precios para la tecnología de Amazon no es tan sencilla como lo que Microsoft proporciona [52].

3. *Watson Machine Learning* [*53*]: crea modelos de *ML* con herramientas de aprendizaje automático visuales que ayudan a los usuarios a detectar patrones y tomar decisiones aplicando *Deep Learning*.

   Usa herramientas de ciencia de datos de código abierto e interactua con la información de arrastrar y soltar en los tableros [55].
4. *Google Cloud Machine Learning Engine* [*56*]: permite entrenar modelos de aprendizaje automático y después usar la predicción en línea o la predicción por lotes. sus tarifas son sustancialmente menores que otros proveedores [57].
5. *BigML* [*58*]: se basa en el aprendizaje automático [*59*]. Importa datos de múltiples fuentes y construye modelos rápidamente. También se puede insertar en aplicaciones móviles y usarlos para hacer predicciones. Ofrece un plan gratuito, aunque también dispone de niveles premium que son de pago (a partir de los 30$) [*60*].
6. *Dataiku* [*61*]: ofrece bibliotecas de aprendizaje automático usando *Python*. Dispone de una interfaz donde se puede realizar el entrenamiento de cualquier modelo. No publica los precios de su servicio ya que no están en su página web, pero estos pueden variar según las necesidades.
7. *Weka* [*62*]: programa especializado en minería de datos y aprendizaje automático capaz de importar datos, filtrarlos y utilizarlos para crear modelos, entre otras opciones. Dispone de una cantidad elevada de herramientas y de gran utilidad. Creada con *Java* y distribuida de forma libre, junto con su *API* y documentación.

Entre algunas de sus características se encuentran visualizadores de datos, tablas y agrupamientos de datos respecto a la actividad desarrollándose en el propio programa, un catálogo de filtros para los datos, su propia extensión de archivos que *Weka* reconoce automáticamente (*arff*), una

consola por donde ejecutar órdenes y una gran gama de clasificadores y estructuras de datos como se muestra en la Figura 3.10.

La minería de datos o *Data Mining* utiliza cuatro clases de tareas:

- Clasificación.
- Agrupamiento (*clustering*).
- Regresión.
- Reglas de asociación.

Para llevarlas a cabo, se sirve de técnicas estadísticas, algoritmos matemáticos y algoritmos de aprendizaje automático que ayudan a mejorar el rendimiento en base a la experiencia.

También proporciona acceso a bases de datos vía *SQL* gracias a la conexión *Java Database Connectivity* (*JDBC*) y puede procesar el resultado devuelto por una consulta hecha a la base de datos.

Figura 3.10. Interfaz Weka escritorio

Escogemos Weka, debido a su gran sencillez, a que dispone de una interfaz de escritorio y a que reúne numerosos filtros y modelos de entrenamiento que ayudan a valorar con gran detalle el banco de datos a obtener.

El estudio de los otros 2 (Google Cloud e IBM Watson Machine Learning) quedan descritos en el Anexo C.

3.6 CSV y formato de archivo ARFF

Weka trabaja con archivos de extensión ARFF que son de texto plano y se compone de dos partes: la cabecera y los datos.

Archivos ARFF

Antes de empezar a comentar ejemplos que hemos realizado con nuestros archivos analizamos los archivos Weka:

1. *La cabecera:* incluye la definición del nombre de la relación y la declaración de los atributos y su tipo.
    - *Definición de relación*: define el nombre de la relación de datos que contiene el archivo (Figura 3.11).

      El nombre de la relación debe ser de tipo String.

```
@relation <nombre_de_la_relacion> <tipo_de_dato>
```

Figura 3.11. Definición de la relación

```
@attribute <nombre_del_atributo> <tipo_de_dato>
```

Figura 3.12. Declaración de los atributos

- *Declaración de atributos*: después de definir la relación, se deben declarar los atributos de los datos y el tipo de valor que admite cada uno, un ejemplo se muestra en la Figura 3.12. La elección de los atributos que deben aparecer en la tabla se toma en un proceso previo de selección de datos. El formato de declaración de un atributo depende del tipo de dato.

  Los tipos de datos que aceptan estos archivos y algunos ejemplos son:

  - *Numéricos*: se definen como NUMERIC, REAL o INTEGER para números (Figura 3.13).
  - *Fechas*: se escribe el dato de la fecha seguido de DATE (Figura 3.14).
  - También se pueden hacer combinaciones con las distintas unidades de tiempo (Figura 3.15).
  - *Cadenas de caracteres*: se declaran como STRING y deben seguir las normas básicas comentadas al principio sobre las separaciones de espacios (Figura 3.16).
  - *Enumerados*: definir atributos y dar una lista de los posibles valores que puede tomar en nuestra relación (Figura 3.17). Estos valores se deben expresar entre corchetes y separados entre sí por comas.

  Una utilidad práctica es usar una declaración enumerado para suplir la falta del tipo boolean (Figura 3.18).

```
@attribute numero NUMERIC
@attribute numero_real REAL
@attribute numero_entero INTEGER
```

Figura 3.13. Datos numéricos en un archivo ARFF

```
@attribute fecha_completa DATE 'dd-MM-yyyy HH:mm:ss'
```

Figura 3.14. Fechas en un archivo ARFF

```
@attribute hora DATE HH:mm
@attribute anyo DATE yyyy
@attribute fichaje DATE 'dd-MM HH:mm'
```

Figura 3.15. Unidades de Tiempo en ARFF

```
@attribute 'cadena de caracteres' STRING
@attribute cadena_de_caracteres STRING
```

Figura 3.16. Cadenas de Caracteres en ARFF

```
@attribute naipe {As, 2, 3, 4, 5, 6, 7, Sota, Caballo, Rey}
@attribute 'Sistema operativo' {Windows, 'Distro Linux', 'Mac OS'}
```

Figura 3.17. Listas de valores en un documento ARFF

```
@attribute boolean {TRUE, FALSE}
```

Figura 3.18. Atributos Booleanos en un documento ARFF

```
@data
```

Figura 3.19. Se abre el apartado de los datos en un ARFF

2. *Datos* (*datasets*): la sección de los datos comienza con la declaración, que es una simple línea que denota el comienzo, Figura 3.19.
   Seguida de esta declaración, se encuentran los datos. Cada línea representa una instancia. Dentro de cada instancia, los atributos se separan con comas. Estos atributos deben aparecer en el orden en que se declaran en la sección de la cabecera.
   Si algún valor no está definido, se representan con el signo "?". Por último, también podemos atender a la definición de la clase.

   - *Atributo clase o de referencia:* la etiqueta de clase se asigna de manera simbólica a uno de los atributos de la relación (por defecto, será el último de la declaración de atributos). El atributo clase o de referencia se convierte así en la variable que deseamos predecir, pero no se distingue de los demás en el archivo. ARFF. Así, podemos cambiar la etiqueta de clase cuantas veces deseemos. La importancia de esta etiqueta se puede observar más claramente en los procesos de clasificación, ya que son interesantes para construir los modelos (llamados también clasificadores).

4. Sistema desarrollado

En este capítulo se muestra la implementación del sistema desarrollado.

4.1 Introducción

En este TFG se empezó a trabajar por la línea de órdenes o ventana de terminal, en un sistema operativo *MacOS*. En ese equipo comenzamos el renderizado de los PDF desde el terminal logrando conseguirlo con éxito, así como el OCR correspondiente, ante el gran número de documentos de entrada y la posibilidad de agilizar todo este proceso se decidió realizar la implementación en *Python* usando la herramienta *PyCharm* [63], logrando así aligerar todos estos procesos y también el de la obtención de vectores identificativos. Sin embargo, en la parte final del proyecto, debido a que la pandemia de COVID-19 nos impidió seguir haciendo uso de ese equipo se cambió de sistema operativo a *Windows*, debido a que desde el inicio se buscó en todo momento el uso de bibliotecas que puedan usarse en ambos sistemas operativos, no existió ningún tipo de conflicto en ese cambio. Realizando la última parte del proyecto que corresponde al uso de la IA en *Windows* únicamente como describimos en la Figura 4.1. En esta figura queda descrito como se trabajan los 3 primeros pasos en *PyCharm* en *MacOS* y luego tras lo

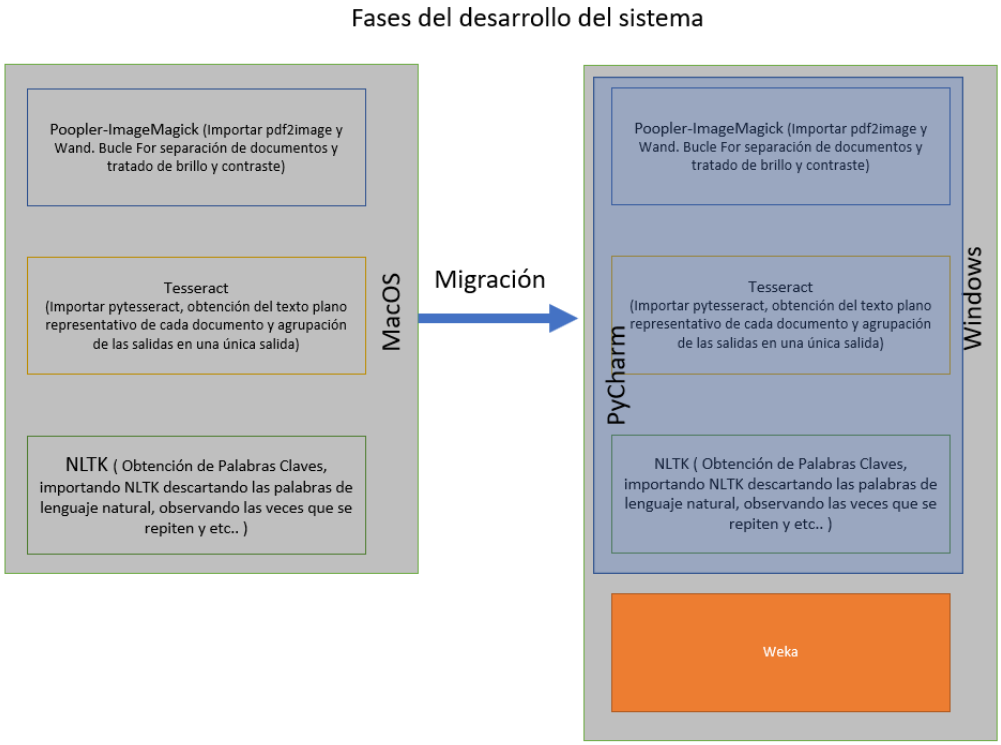


Figura 4.1. Fases del desarrollo entre los 2 Sistemas Operativos

explicado se realiza la migración a Windows para acabar realizando el último paso de la IA en este último SO.

En lo que se refiere a la instalación en *MacOS* primero descargamos la biblioteca poppler y luego realizamos la instalación:

- `ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" 2> /dev/null`
- `brew install poppler.`

En el caso de *Windows* haciendo uso de la orden *conda* podemos instalarlo también sin mayor tipo de problema.

- *conda install -c conda-forge poppler*

En la Figura 4.2. se muestra como aplicamos la orden a nuestros PDF, en el caso en concreto de uno que se llama *RENOVACION.pdf*, se generan 3 archivos ppm donde el único que tiene información de valor es el *foo-page-002.ppm*.

Si aplicamos la siguiente orden al archivo PDF, se obtendría una imagen por cada página del documento:

- *pdfimages RENOVACION.pdf foo-page*

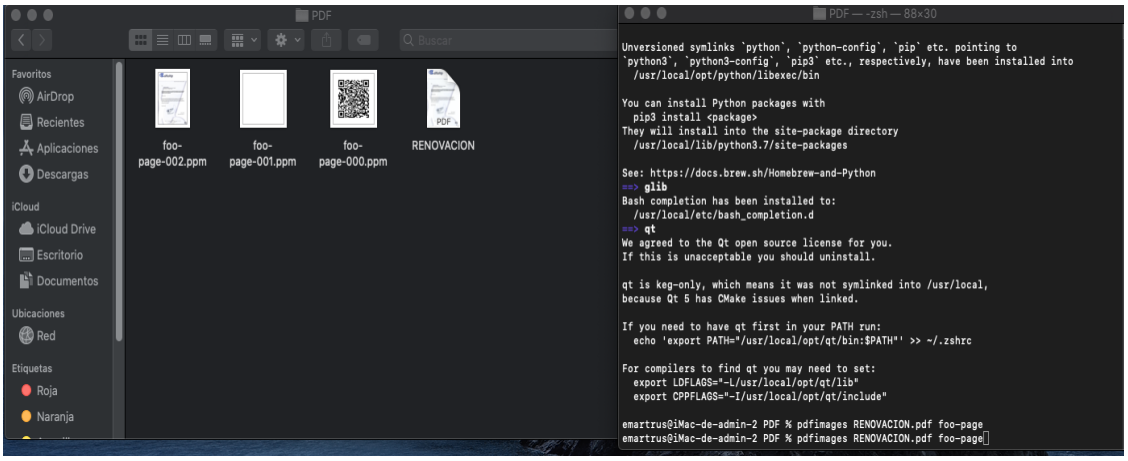


Figura 4.2. Resultado de Poppler en MacOS por Terminal

```
path = "."
#Lista vacia para incluir los ficheros
lstFiles =[]

#Lista con todos los ficheros del directorio:
lstDir = os.walk(path)

#Creamos una lista de los ficheros pdf que existen en el directorio y los incluyo en la lista
for root,dirs,ficheros in lstDir:
    for fichero in ficheros:
        (nombreFichero, extension) = os.path.splitext(fichero)
        if(extension == ".pdf"):
            lstFiles.append(nombreFichero+extension)

print(lstFiles)
print("LISTADO FINALIZADO")
print ("longitud de la lista = ", len(lstFiles))
```

Figura 4.3. Tratamiento de los PDF en Pyhton

En la Figura 4.3 se muestra cómo realizar la implementación en *PyCharm*, a través del uso de bucles *for*, obtenemos la lista de los archivos PDF que tengamos en el directorio donde nos encontramos, destacar que *path* está declarado para estar ubicados en el directorio donde se ejecuta el script.

Con el método *os.walk(path)* obtenemos una lista de todos los archivos que tengamos en el directorio donde estemos, recorriendo esta lista obtenemos dichos archivos, con el *splitext* obtenemos una tupla con el tipo de objeto y su nombre, así pues con el siguiente *if*, obtenemos todos los archivos *.pdf* y los añadimos a un array vacío donde se tendría con su valor.

Tras esto, recorremos el array de PDF uno por uno y separando en las imágenes en primer lugar como se muestra en la Figura 4.4.

Hay que mencionar que con *fname* damos nombre a cada *png* en función del nombre del archivo y con el *shutil* hacemos una copia del archivo a un directorio con el mismo nombre del archivo.

```
for proyect in lstFiles:

    print(proyect)
    nombredirectorio="Directorio"+proyect
    os.mkdir(nombredirectorio)
#En esta parte realizamos el renderizado de las imagenes con Poppler que en Mac se refiere pdf2image
    pages = convert_from_path(proyect)
    #print(pages)

    print(pages)
    print("longitud de paginas: ", len(pages))
#Se crea la clase MyImage donde se definen los pasos a realizar para aumentar brillo  y contraste.
    for i,page in enumerate(pages):
            fname =proyect+'image'+str(i)+'.png'
            print(fname)
            page.save(fname,'PNG')
            shutil.copy(fname, nombredirectorio)
```

Figura 4.4. Tratamiento de los PDF por página y guardando cada página como Imagen

4.2 Renderizado de los documentos y tratado de las imágenes

Para su instalación de *ImageMagick* en *MacOS* utilizamos:

- `ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" < /dev/null 2> /dev/null`

Y posteriormente para instalarlo con *brew*:

- *brew install imagemagick; brew install exiftool; #etc*

En el caso de *Windows* la instalación también es muy sencilla, si entramos en la página web de la biblioteca de *ImageMagick* [64] solo tenemos que instalar el instalador deseado en función del nº de bits de nuestro Sistema Operativo tal y como se muestra en la Figura 4.5.

Install ImageMagick on Windows

You could build ImageMagick by yourself, but it requires a build tool chain like Visual Studio to compile it. The easiest way is simply downloading a prebuilt binary of ImageMagick for your architecture (`win32` or `win64`).

You can download it from the following link:

http://www.imagemagick.org/download/binaries/

Choose a binary for your architecture:

Windows 32-bit
ImageMagick-6.9.0-4-Q16-x86-dll.exe

Windows 64-bit
ImageMagick-6.9.0-4-Q16-x64-dll.exe

Figura 4.5. Instaladores ImageMagick Windows

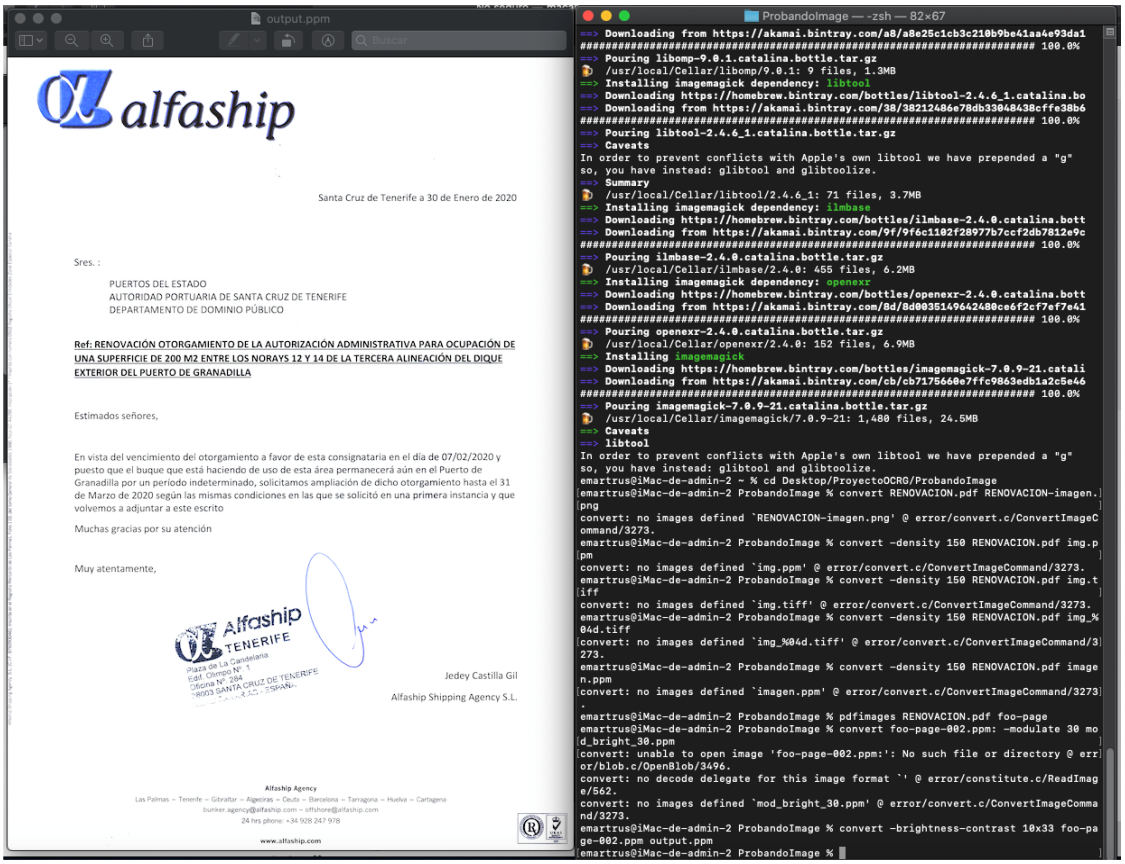


Figura 4.6. Salida de ImageMagick por terminal

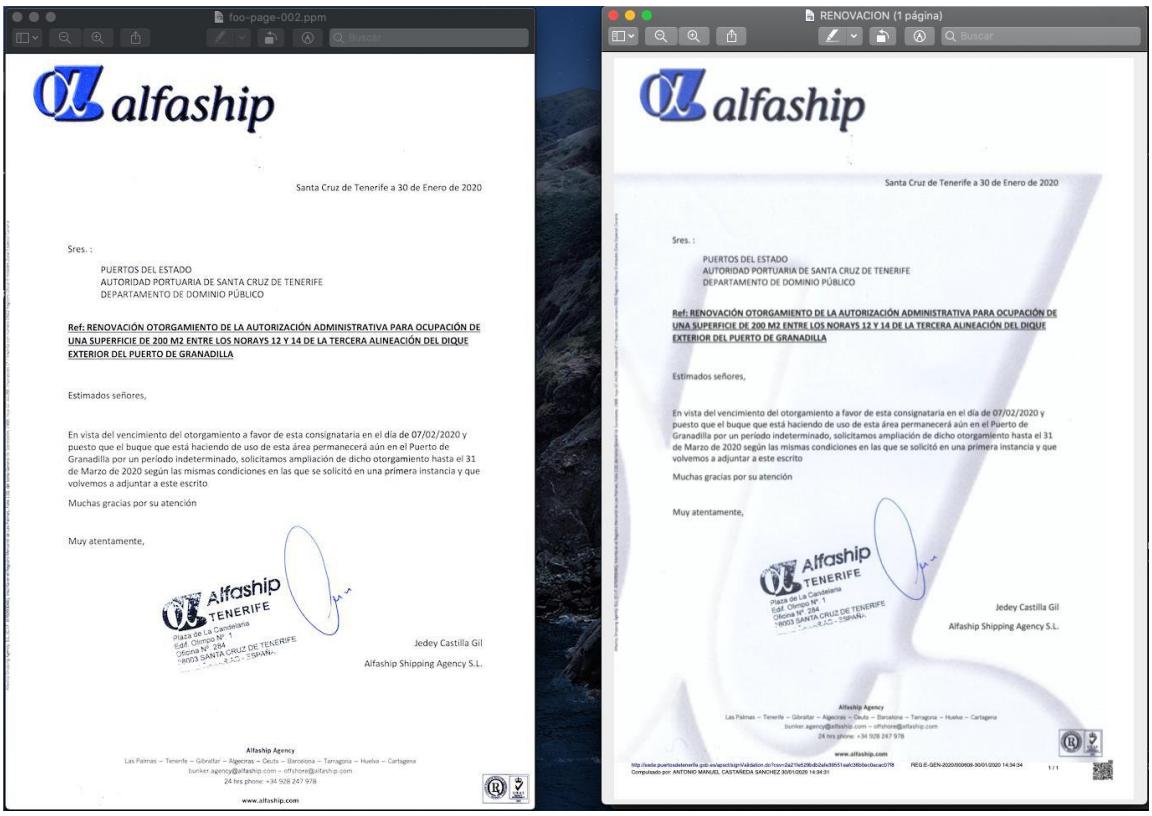


Figura 4.7. Documento original a la Derecha y el resultado con GIMP a la Izquierda

Como comenzamos desde el Terminal de MacOS se muestra como editar las imágenes a través de este ejecutando órdenes como:

- *convert -brightness-contrast 10x33 nombreArchivoEntrada.ppm nombreArchivoSalida.ppm*

En la Figura 4.6. se muestra cómo se editan las imágenes siendo los números 10x33 un indicador del brillo y contraste. Ambos números pueden ir desde -100 a 100, siendo los números negativos una reducción del efecto a editar.

Por tanto, en nuestro ejemplo aumentamos el brillo en un valor de 10 y el contraste en un valor de 33. Comparando el resultado con el documento original podemos observar la mejoría en el tratamiento del documento y viendo cómo se eliminan esos elementos no deseados ya mencionadas en numerosas ocasiones como

```
if i>0:
    for i, page in enumerate(pages):
        imagenEdit = proyect+'imageEdit' + str(i) + '.png'
        fname = proyect+'image' + str(i) + '.png'
        with MyImage(filename=fname) as img:
            img.brightness_contrast(10, 33)
            img.save(filename=imagenEdit)
            shutil.copy(imagenEdit, nombredirectorio)
            os.remove(fname)

else:
    with MyImage(filename=proyect+'image0.png') as img:
        img.brightness_contrast(10, 33)
        img.save(filename=proyect+'imagen0Edit.png')
        shutil.copy(proyect+'imagen0Edit.png',nombredirectorio)
        os.remove(fname)
```

Figura 4.8. Tratamiento de las imágenes en Python

marcas de agua y demás. Si bien el resultado se parece al que conseguimos con *GIMP* la ventaja de *ImageMagick* es que nos permite tratar un gran número de imágenes, mientras que con el *GIMP* perderíamos demasiado tiempo con cada imagen a editar, Figura 4.7.

Tras los pasos de código explicados en el apartado anterior vemos que a continuación procedemos a explicar la implementación del *ImageMagick* en *Pycharm*.

En la Figura 4.8. se muestra el uso de *Image* para aumentar el brillo de las imágenes que nos han salido en función de la página, hay que decir que hemos tenido que contemplar los casos en los que los que los PDF solo tengan una página.

El brillo y el contraste se trata con la función: *brightness_contrast (10,33)*.

Siendo los números 10x33 un indicador del brillo y contraste correspondiente como ya hicimos en la ventana de órdenes.

4.3 Reconocimiento óptico de caracteres

La instalación de *Tesseract* en *MacOS* es muy similar a las anteriores instalaciones, siguiendo los pasos de a continuación:

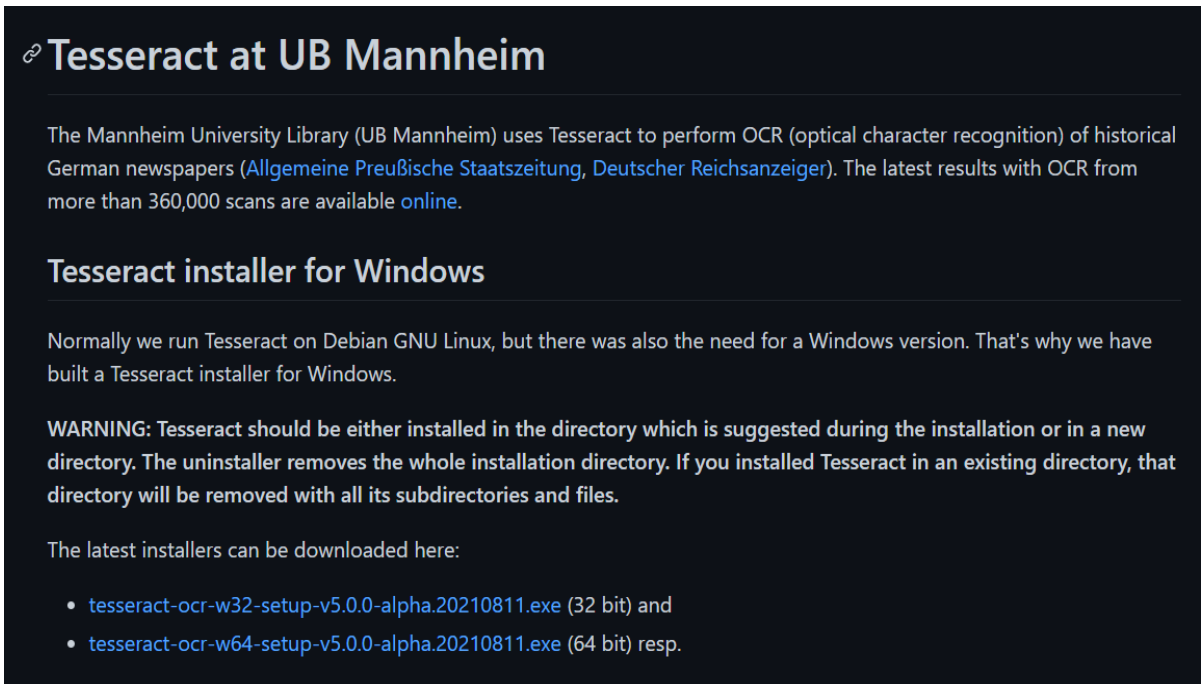


Figura 4.9. Instalación Tesseract Windows

- *ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" < /dev/null 2> /dev/null*

Para posteriormente hacer: *brew install tesseract*

En lo que respecta a la instalación en *Windows* entrando en el *GitHub* de *Tesseract* tenemos los ejecutables para su instalación, Figura 4.9 [62].

Para trabajar desde el terminal en MacOS tenemos que hacer uso de la orden:

- *tesseract imagen.ppm out*

En Pycharm tras tener nuestras imágenes editadas, realizamos el OCR, usando pytesseract, Figura 4.10. Con el método *imagetostring()* pasándole como parámetro de entrada una de las imágenes, generando así *salidas.txt N* siendo *N* el número de páginas del PDF.

Hay que destacar que volvemos a estudiar el caso de que el PDF tenga más de una sola página o únicamente una sola página.

```
#Ahora nos toca realizar OCR.
    if i>0:
        for i, page in enumerate(pages):
            imagenEdit = proyect+'imageEdit' + str(i) + '.png'
        #Cargamos la imagen utilizando opencv
            imagen = cv2.imread(filename=imagenEdit)
        #Extraemos texto de la imagen
            sret = pytesseract.image_to_string(imagen)
        #Mostramos el resultado
            text = proyect +'salida' + str(i) + '.txt'
            f = open(text,'w')
            f.write(sret)
            f.close()
            shutil.copy(text,nombredirectorio)
            os.remove(imagenEdit)

    else:
        imagen = cv2.imread(filename=proyect+'imagen0Edit.png')
        sret = pytesseract.image_to_string(imagen)
        #Mostramos el resultado
        text = proyect+'salida0.txt'
        f = open(text,'w')
        f.write(sret)
        f.close()
        shutil.copy(text,nombredirectorio)
        os.remove(proyect+'imagen0Edit.png')
```

Figura 4.10. Implementación del OCR en Python

```
    files = []
    for i, page in enumerate(pages):
        text = proyect+'salida' + str(i) + '.txt'
#Buscaremos listar las palabras de nuestro pdf que ha sido convertido en txt
        files.append(text)
        with open(proyect+'salidaUnida.txt', 'w') as outfile:
            for unido in files:
                with open(unido) as infile:
                    for line in infile:
                        if line:
                            outfile.write(line)
```

Figura 4.11. Formado de una única salida de Texto Plano

Para obtener un único diccionario, necesitamos tener un único *txt* por *PDF*, por lo tanto, decidimos concatenar los distintos archivos para obtener un único *txt* tal y como muestra en la Figura 4.11, del que obtenemos las palabras significativas y una nube de tags.

4.4 Obtención de vectores identificativos

La instalación de la biblioteca de lenguaje natural que ya hemos explicado en el capítulo 3.4 se realiza de igual manera tanto en *Windows* como en *MacOS*. El proyecto a partir de este paso se realiza únicamente en *Pycharm* a excepción de *Weka*.

A continuación, explicamos con más detalle una de las partes más tediosas del proyecto como fue la de cómo confeccionar un diccionario y el desarrollo de las diferentes opciones como pueden bigramas-trigramas o términos simples.

4.4.1 Detección de palabras sin preprocesado

En la Figura 4.12 se muestra cómo hacemos el procesado de las palabras utilizando el metodo *split()* que devuelve una lista de palabras o tokens usando un *sep* como separador. Además, podemos especificar el separador particular (el separador predeterminado es cualquier espacio en blanco.

4.4.2 Método split para contar candidatos a palabras

A continuación, importamos el método que crea un diccionario con la frecuencia de aparición de los elementos en una lista:

- *from collections import Counter*

Tras esto, usamos el método *Counter* para contar el número de veces que los candidatos a palabra aparecen en el texto:

- *print("CÁLCULO MÉTODO SPLIT:", Counter(word_candidates), "\n")*

Seguimos con una relación significante-significado donde a la lista de candidatos se les quita los símbolos '"' y '.', Figura 4.13.

- *strippedcandidates = [w.strip('".') for w in wordcandidates]*

Le podemos añadir otros símbolos (por ejemplo: *?*, *!*, *:*...) y después mostramos el número de veces que los candidatos aparecen en el texto tras sacarles los signos de puntuación:

```
['ja', '!', 'Canaria', 'Alfaship', 'Shipping', 'Agency', ',', 'S.L', '.', '(', '
C.1.F', '.', 'B76093046', ')', ',', 'Inscrita', 'en', 'el', 'Registro', 'Mercant
il', 'de', 'Las', 'Paimas', ',', 'Folio', '110', ',', 'del', 'tomo', 'Genera', '
!', 'de', 'Sociedades', '1988', ',', 'hoja', 'GC-44288', '.', 'Inscripción', '1º
', '/', 'Inscrita', 'con', 'numero', '0502', 'Registro', 'Oficial', 'Entidades',
 'Zona', 'Especi', ':', 'OU', 'alfaship', 'Santa', 'Cruz', 'de', 'Tenerife', 'a'
, '30', 'de', 'Enero', 'de', '2020', 'Sres', '.', ':', 'PUERTOS', 'DEL', 'ESTADO
', 'AUTORIDAD', 'PORTUARIA', 'DE', 'SANTA', 'CRUZ', 'DE', 'TENERIFE', 'DEPARTAME
NTO', 'DE', 'DOMINIO', 'PUBLICO', 'Ref', ':', 'RENOVACION', 'OTORGAMIENTO', 'DE'
, 'LA', 'AUTORIZACION', 'ADMINISTRATIVA', 'PARA', 'OCUPACION', 'DE', 'UNA', 'SUP
ERFICIE', 'DE', '200', 'M2', 'ENTRE', 'LOS', 'NORAYS', '12', 'Y', '14', 'DE', 'L
A', 'TERCERA', 'ALINEACION', 'DEL', 'DIQUE', 'EXTERIOR', 'DEL', 'PUERTO', 'DE',
'GRANADILLA', 'Estimados', 'sefiores', ',', 'En', 'vista', 'del', 'vencimiento',
 'del', 'otorgamiento', 'a', 'favor', 'de', 'esta', 'consignataria', 'en', 'el',
 'dia', 'de', '07/02/2020', 'y', 'puesto', 'que', 'el', 'buque', 'que', 'esta',
'haciendo', 'de', 'uso', 'de', 'esta', 'drea', 'permanecera', 'aun', 'en', 'el',
 'Puerto', 'de', 'Granadilla', 'por', 'un', 'periodo', 'indeterminado', ',', 'so
licitamos', 'ampliación', 'de', 'dicho', 'otorgamiento', 'hasta', 'el', '31', 'd
e', 'Marzo', 'de', '2020', 'segun', 'las', 'mismas', 'condiciones', 'en', 'las',
 'que', 'se', 'solicit6', 'en', 'una', 'primera', 'instancia', 'y', 'que', 'volv
emos', 'a', 'adjuntar', 'a', 'este', 'escrito', 'Muchas', 'gracias', 'por', 'su'
, 'atencion', 'Muy', 'atentamente', ',', '4', 'oe', 'TENERIE', 'Jedey', 'Castill
a', 'Gil', 'Alfaship', 'Shipping', 'Agency', 'S.L', '.', 'Alfaship', 'Agency', '
Las', 'Palmas', '-', 'Tenerife', '-', 'Gibraltar', '-', 'Algeciras', '-', 'Ceuta
', '-', 'Barcelona', '-', 'Tarragona', '-', 'Huelva', '-', 'Cartagena', 'bunker.
agency', '@', 'alfaship.com', '-', 'offshore', '@', 'alfaship.com', '24', 'hrs',
 'phone', ':', '+34', '928', '247', '978', 'www.alfaship.com']
emartrus@iMac-de-admin-2 prueba %
```

Figura 4.12. Print del metodo Split

```
CALCULO MÉTODO SPLIT:  Counter({'de': 13, '-': 9, 'DE': 8, 'en': 5, 'el': 5, 'a'
: 4, 'que': 4, 'Alfaship': 3, 'del': 3, 'DEL': 3, 'esta': 3, 'Shipping': 2, 'S.L
.': 2, 'Inscrita': 2, 'Registro': 2, 'Las': 2, 'Tenerife': 2, '2020': 2, 'LA': 2
, 'otorgamiento': 2, 'y': 2, 'por': 2, 'las': 2, 'Agency': 2, 'ja!': 1, 'Canaria
': 1, 'Agency,': 1, '(C.1.F.': 1, 'B76093046),': 1, 'Mercantil': 1, 'Paimas,': 1
, 'Folio': 1, '110,': 1, 'tomo': 1, 'Genera!': 1, 'Sociedades': 1, '1988,': 1, '
hoja': 1, 'GC-44288.': 1, 'Inscripción': 1, '1º': 1, '/': 1, 'con': 1, 'numero':
 1, '0502': 1, 'Oficial': 1, 'Entidades': 1, 'Zona': 1, 'Especi:': 1, 'OU': 1, '
alfaship': 1, 'Santa': 1, 'Cruz': 1, '30': 1, 'Enero': 1, 'Sres.': 1, ':': 1, 'P
UERTOS': 1, 'ESTADO': 1, 'AUTORIDAD': 1, 'PORTUARIA': 1, 'SANTA': 1, 'CRUZ': 1,
'TENERIFE': 1, 'DEPARTAMENTO': 1, 'DOMINIO': 1, 'PUBLICO': 1, 'Ref:': 1, 'RENOVA
CION': 1, 'OTORGAMIENTO': 1, 'AUTORIZACION': 1, 'ADMINISTRATIVA': 1, 'PARA': 1,
'OCUPACION': 1, 'UNA': 1, 'SUPERFICIE': 1, '200': 1, 'M2': 1, 'ENTRE': 1, 'LOS':
 1, 'NORAYS': 1, '12': 1, 'Y': 1, '14': 1, 'TERCERA': 1, 'ALINEACION': 1, 'DIQUE
': 1, 'EXTERIOR': 1, 'PUERTO': 1, 'GRANADILLA': 1, 'Estimados': 1, 'sefiores,':
1, 'En': 1, 'vista': 1, 'vencimiento': 1, 'favor': 1, 'consignataria': 1, 'dia':
 1, '07/02/2020': 1, 'puesto': 1, 'buque': 1, 'haciendo': 1, 'uso': 1, 'drea': 1
, 'permanecera': 1, 'aun': 1, 'Puerto': 1, 'Granadilla': 1, 'un': 1, 'periodo':
1, 'indeterminado,': 1, 'solicitamos': 1, 'ampliación': 1, 'dicho': 1, 'hasta':
1, '31': 1, 'Marzo': 1, 'segun': 1, 'mismas': 1, 'condiciones': 1, 'se': 1, 'sol
icit6': 1, 'una': 1, 'primera': 1, 'instancia': 1, 'volvemos': 1, 'adjuntar': 1,
 'este': 1, 'escrito': 1, 'Muchas': 1, 'gracias': 1, 'su': 1, 'atencion': 1, 'Mu
y': 1, 'atentamente,': 1, '4': 1, 'oe': 1, 'TENERIE': 1, 'Jedey': 1, 'Castilla':
 1, 'Gil': 1, 'Palmas': 1, 'Gibraltar': 1, 'Algeciras': 1, 'Ceuta': 1, 'Barcelon
a': 1, 'Tarragona': 1, 'Huelva': 1, 'Cartagena': 1, 'bunker.agency@alfaship.com'
: 1, 'offshore@alfaship.com': 1, '24': 1, 'hrs': 1, 'phone:': 1, '+34': 1, '928'
: 1, '247': 1, '978': 1, 'www.alfaship.com': 1})
```

Figura 4.13. Método Split con el uso de Counter

Figura 4.14. Método Significante-Significado

- *print("CÁLCULO MÉTODO SIGNIFICANTE-SIGNIFICADO:",*
- *Counter(stripped_candidates), "\n")*

Y por último establecemos una relación significante-significado tras convertir las palabras a minúsculas como se muestra en la Figura 4.14.

Creamos una lista de candidatos sin signos de puntuación convertidos a minúsculas *(lower()):*

- *lowercasestrippedcandidates = [s.lower() for s in strippedcandidates]*

Y se muestra el número de veces que los candidatos sin signos de puntuación y en minúsculas aparecen en el texto, como se muestra en la Figura 4.15.

- *print("CÁLCULO TRAS CONVERSIÓN A MINÚSCULAS:",*
  Counter(lowercasestripped_candidates), "\n")
-

4.4.3 Detección de palabras con preprocesado del texto

Importamos las bibliotecas que mostramos arriba para obtener los tokens. Que son la lista de tokens resultado de aplicar el *tokenizer* del *NLTK* al nuestro texto.

- *tokens = [w for w in word_tokenize(headline)]*
- *print(tokens)*

A continuación, se muestra la salida que nos puede dar este último *Token*.

```
'+34': 1, '928': 1, '247': 1, '978': 1, 'www.alfaship.com': 1})

CALCULO TRAS CONVERSION A MINUSCULAS:  Counter({'de': 21, '—': 9, 'en': 6, 'del'
: 6, 'el': 5, 'alfaship': 4, 'las': 4, 'a': 4, 'que': 4, 'tenerife': 3, 'otorgam
iento': 3, 'y': 3, 'esta': 3, 'shipping': 2, 's.l': 2, 'inscrita': 2, 'registro'
: 2, 'santa': 2, 'cruz': 2, '2020': 2, 'la': 2, 'una': 2, 'puerto': 2, 'granadil
la': 2, 'por': 2, 'agency': 2, 'ja!': 1, 'canaria': 1, 'agency,': 1, '(c.1.f': 1
, 'b76093046),': 1, 'mercantil': 1, 'paimas,': 1, 'folio': 1, '110,': 1, 'tomo':
 1, 'genera!': 1, 'sociedades': 1, '1988,': 1, 'hoja': 1, 'gc-44288': 1, 'inscri
pción': 1, '1°': 1, '/': 1, 'con': 1, 'numero': 1, '0502': 1, 'oficial': 1, 'ent
idades': 1, 'zona': 1, 'especi:': 1, 'ou': 1, '30': 1, 'enero': 1, 'sres': 1, ':
': 1, 'puertos': 1, 'estado': 1, 'autoridad': 1, 'portuaria': 1, 'departamento':
 1, 'dominio': 1, 'publico': 1, 'ref:': 1, 'renovacion': 1, 'autorizacion': 1, '
administrativa': 1, 'para': 1, 'ocupacion': 1, 'superficie': 1, '200': 1, 'm2':
1, 'entre': 1, 'los': 1, 'norays': 1, '12': 1, '14': 1, 'tercera': 1, 'alineacio
n': 1, 'dique': 1, 'exterior': 1, 'estimados': 1, 'sefiores,': 1, 'vista': 1, 'v
encimiento': 1, 'favor': 1, 'consignataria': 1, 'dia': 1, '07/02/2020': 1, 'pues
to': 1, 'buque': 1, 'haciendo': 1, 'uso': 1, 'drea': 1, 'permanecera': 1, 'aun':
 1, 'un': 1, 'periodo': 1, 'indeterminado,': 1, 'solicitamos': 1, 'ampliación':
1, 'dicho': 1, 'hasta': 1, '31': 1, 'marzo': 1, 'segun': 1, 'mismas': 1, 'condic
iones': 1, 'se': 1, 'solicit6': 1, 'primera': 1, 'instancia': 1, 'volvemos': 1,
'adjuntar': 1, 'este': 1, 'escrito': 1, 'muchas': 1, 'gracias': 1, 'su': 1, 'ate
ncion': 1, 'muy': 1, 'atentamente,': 1, '4': 1, 'oe': 1, 'tenerie': 1, 'jedey':
1, 'castilla': 1, 'gil': 1, 'palmas': 1, 'gibraltar': 1, 'algeciras': 1, 'ceuta'
: 1, 'barcelona': 1, 'tarragona': 1, 'huelva': 1, 'cartagena': 1, 'bunker.agency
@alfaship.com': 1, 'offshore@alfaship.com': 1, '24': 1, 'hrs': 1, 'phone:': 1, '
+34': 1, '928': 1, '247': 1, '978': 1, 'www.alfaship.com': 1})
```

Figura 4.15. Con la conversión a minúscula

```
    textoUnido = open(proyect+'salidaUnida.txt', 'r')
    headline = textoUnido.read()
    word_candidates = headline.split()
    #nltk.download(), se ha de ejecutar siempre la primera vez que se ejecuta
#Número de veces que los candidatos a palabra aparecen en el texto
    print("CALCULO MÉTODO SPLIT: ", Counter(word_candidates), "\n")

#Creamos una lista de candidatos a los cuales les quitamos los símbolos '"' y '.'
    stripped_candidates = [w.strip('".') for w in word_candidates]

#Número de veces que los candidatos aparecen en el texto sin signos como puntos de puntuación
    print("CALCULO METODO SIGNIFICANTE-SIGNIFICADO: ", Counter(stripped_candidates), "\n")

#Ahora pasaremos el titulo a minuscula y crearemos una lista de candidatos sin los signos como antes y
    lower_case_stripped_candidates = [s.lower() for s in stripped_candidates]
    print("CALCULO TRAS CONVERSION A MINUSCULAS: ", Counter(lower_case_stripped_candidates), "\n")
```

Figura 4.16. Implementación de los Splits en Python

4.4.4 Obtención de la lista de tokens con un tokenizer y filtrado no alfabético

De la lista de candidatos, queremos quitar los candidatos que no tienen caracteres alfabéticos ('"', '.'...). Para ello, importamos la biblioteca de *Python* encargada de la gestión de expresiones regulares:

- *import re*

Obtenemos una lista de tokens del texto que han pasado el filtro. Para obtener la lista, como se muestra en la Figura 4.17:

1. Se convierte el texto a minúsculas (*headline.lower()*)
2. Se tokeniza el texto en minúsculas (*wordtokenize(headline.lower())*)
3. Se añaden a la lista alphatokens los tokens que empiezan por un carácter alfabético (*re.match("^[a-z]+.*)*)
    - *alphatokens = [w for w in wordtokenize(headline.lower()) if re.match("^[a-z]+.", w)]**
    - *print(alpha_tokens)*

4.4.5 PoS tagging

Tras realizar los *imports* procedemos a través del *Pos* a etiquetar las categorías gramaticales de cada palabra del texto como mostramos en la Figura 4.19.

#Lista de tokens del texto convertido en minúsculas

- *tokens = [w for w in word_tokenize(headline.lower())]*

#PoS tagging de los tokens de la lista

- *tagged_tokens = nltk.pos_tag(tokens)*

Todas las etiquetas las se muestran en la Figura 4.20.

Figura 4.17. Salidas de los tokens

```
#alpha_tokens es la lista de tokens del texto que han pasado el filtro. Para obtener la lista:
#1. Se convierte el texto a minúsculas (headline.lower(e
#2. Se tokeniza el texto en minúsculas (word_tokenize(headline.lower()))
#3. Se añaden a la lista alpha_tokens los tokens que empiezan por un caracter alfabético (re.match("^[a-z]+

    alpha_tokens = [w for w in word_tokenize(headline.lower()) if re.match("^[a-z]+.*", w)]
    print("Mostramos alpha_tokens")
    print(alpha_tokens)
```

Figura 4.18. Aplicar NLTK en Python

```
#Para hacer el etiquetaje de categoría gramatical de un texto, importamos el tokenizer (word_tokenize) y
#el PoS tagger (pos_tag), ambos de la librería NLTK
from nltk import word_tokenize, pos_tag
```

Figura 4.19. Biblioteca NLTK

Si queremos ver que significan estas etiquetas *PoS* podemos describir su significado:

- *DT*: Determinante.
- *JJ*: Adjetivo.
- *NN*: Nombre en singular.
- *NNS*: Nombre en plural.
- *VBD*: Verbo en pasado.
- *VBG*: Verbo en gerundio.
- *MD*: Verbo modal.
- *IN*: Preposición.
- *PRP*: Pronombre.
- *RB*: Adverbio.
- *CC*: Conjunción coordinada.
- *CD*: Numeral.

4.4.6 Buscador de n-gramas

#Lista de tokens del texto convertido en minúsculas

- *tokens = [w for w in word_tokenize(headline.lower())]*

#Lista de bigramas

- *print ("BIGRAMAS:", list(ngrams(tokens, 2)),"\n")*

#Lista de trigramas

- *print ("TRIGRAMAS:", list(ngrams(tokens, 3)))*

Una vez importamos todo lo que necesitamos, podemos obtener diferentes n-gramas que nos puedan interesar como puedan ser correos electrónicos, números de teléfonos y demás (Figura 4.22). Pero no todos los ngramas nos pueden interesar.

Figura 4.20. Etiquetado POS en Python

```
#Para hacer la búsqueda de n-gramas de un texto, importamos el tokenizer (word_tokenize) y los métodos de búsqueda
#de n-gramas, ambos de la librería NLTK

from nltk import word_tokenize
from nltk.util import ngrams
```

Figura 4.21. Import de los ngramas

4.4.7 Búsqueda de n-gramas que son colocaciones

Para hacer la búsqueda de n-*gramas* que son colocaciones, importamos el *tokenizer* (*word_tokenize*) y los métodos de búsqueda de colocaciones, ambos de la biblioteca *NLTK*.

- *from nltk import word_tokenize*
- *from nltk.collocations import **

De los métodos de búsqueda de colocaciones, cargamos las métricas de cálculo de los bigramas y trigramas candidatos a ser colocaciones como se muestra en la Figura 4.22.

- *bigram_measures = nltk.collocations.BigramAssocMeasures()*
- *trigram_measures = nltk.collocations.TrigramAssocMeasures()*

Definimos las funciones a utilizar como pueden ser *getcollcandidates(), refiltercandidates()* y *getnbest_candidates()*.

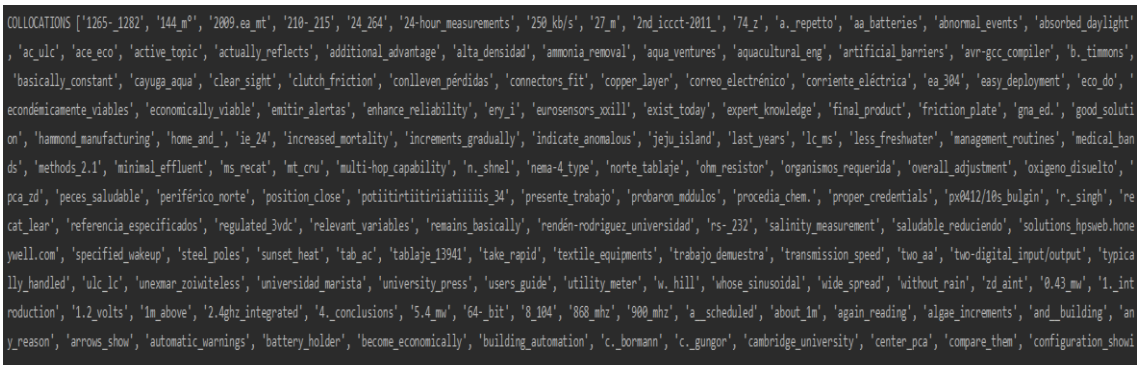


Figura 4.22. Bigramas y Trigramas salida Python

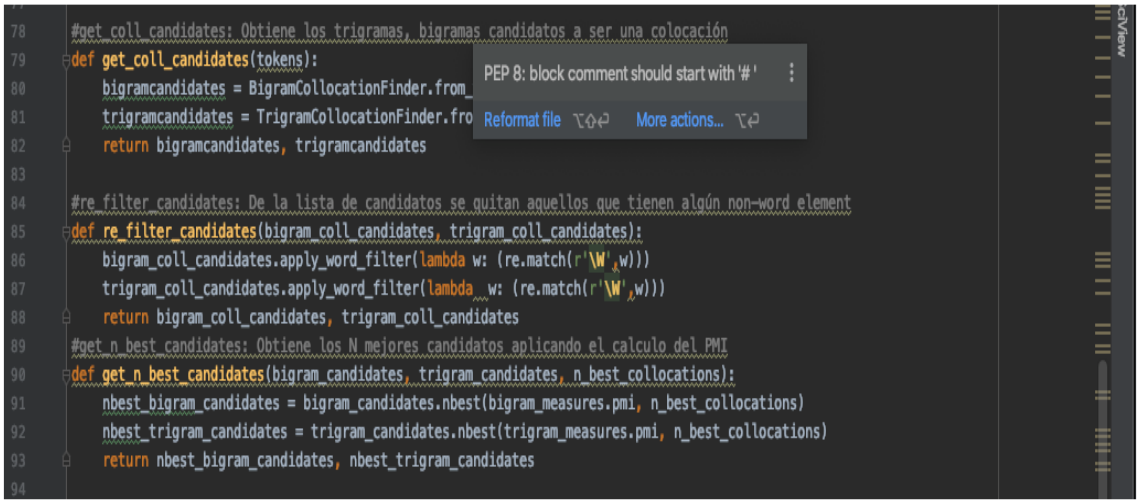


Figura 4.23. implementación de los métodos de arriba

Figura 4.24. Función de GetCollocations

Donde:

1. *getcollcandidates()* recibe de parámetro de entrada los tokens que mencionamos antes y obtiene los trigramas y bigramas candidatos a ser una colocación.
2. *refiltercandidates()* recibe de entrada tanto los bigramas como trigramas que sean candidatos y se quitan aquellos que tengan algún non-word element.*getnbest_candidates()* recibe parámetros de entrada tanto los bigramas, como trigramas como las mejores colocaciones, y obtiene por salida los mejores candidatos.

Y por último, la función *get_collocations* realiza las siguientes acciones principales:

1. Crea la lista de tokens del texto (*word_tokenize*).
2. A partir de la lista de tokens, busca los *bigramas* y *trigramas* que son candidatos a ser colocaciones según unos cálculos estadísticos. En este caso, se realiza el cálculo del *Pointwise Mutual Information (PMI)*.
3. Retorna los N mejores candidatos.

   Dando por salida los mejores bigramas y trigramas candidatos.

```
TERMINOS CANDIDATOS A SER COLOCACIONES ['200_m2', '24_hrs', '247_978', '4_oe',
 '24_hrs_phone', '247_978_www.alfaship.com', '4_oe_tenerie', '928_247_978', 'a
emartrus@iMac-de-admin-2 prueba %
```

Figura 4.25. Ejemplo de Trigramas obtenidas

```
#creacion del word cloud


    word2display=" ".join([lu.replace(' ','_') for lu in collocations])
    wordcloud = WordCloud(background_color='white', max_font_size=500).generate(word2display)
    wordcloud.to_file(proyect+'salidaNube.png')
    shutil.copy(proyect + 'salidaNube.png', nombredirectorio)
    os.remove(proyect+'salidaNube.png')
    plt.figure()
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    for i, page in enumerate(pages):
        borrador = proyect + 'salida' + str(i) + '.txt'
        os.remove(borrador)
    print("SACAMOS LOS VALORES DE ESTA COSA", proyect+'salidaUnida.txt')
    shutil.copy(proyect + 'salidaUnida.txt', nombredirectorio)
    os.remove(proyect+'salidaUnida.txt')
```

Figura 4.26. Implementación de la Nube de Tags en Python

Una vez tenemos esto, se obtiene la nube de tags tanto de esta manera como con *PyTagCloud*, como con esta última lo hemos conseguido de una manera mucho más rápida, aplicamos esta, pero hemos de destacar que de la otra manera podemos realizar un filtrado con mayor personalización.

4.4.8 Nube de Tags

Por último, en nuestro código se muestra cómo se obtiene la nube de palabras o de tags y como se guarda en la Figura 4.26.

Destacar que todo documento que se genera se usa en un método o un paso posterior, por eso una vez que los obtenemos los vamos copiando en nuestro directorio y se borra del directorio raíz, es por ello que se incorporan los métodos *os.remove()*.

Como se muestra en la Figura 4.27, nuestro proyecto funciona adecuadamente. Destacar que por ejemplo para el *pdf documentoAsiento-8* tenemos 4 páginas y por tanto primero salen 4 imágenes, después se convierten en otras 4 editadas, se obtiene el OCR de cada una de ellas en formato *txt* y una salida de las palabras claves en formato *PNG*.

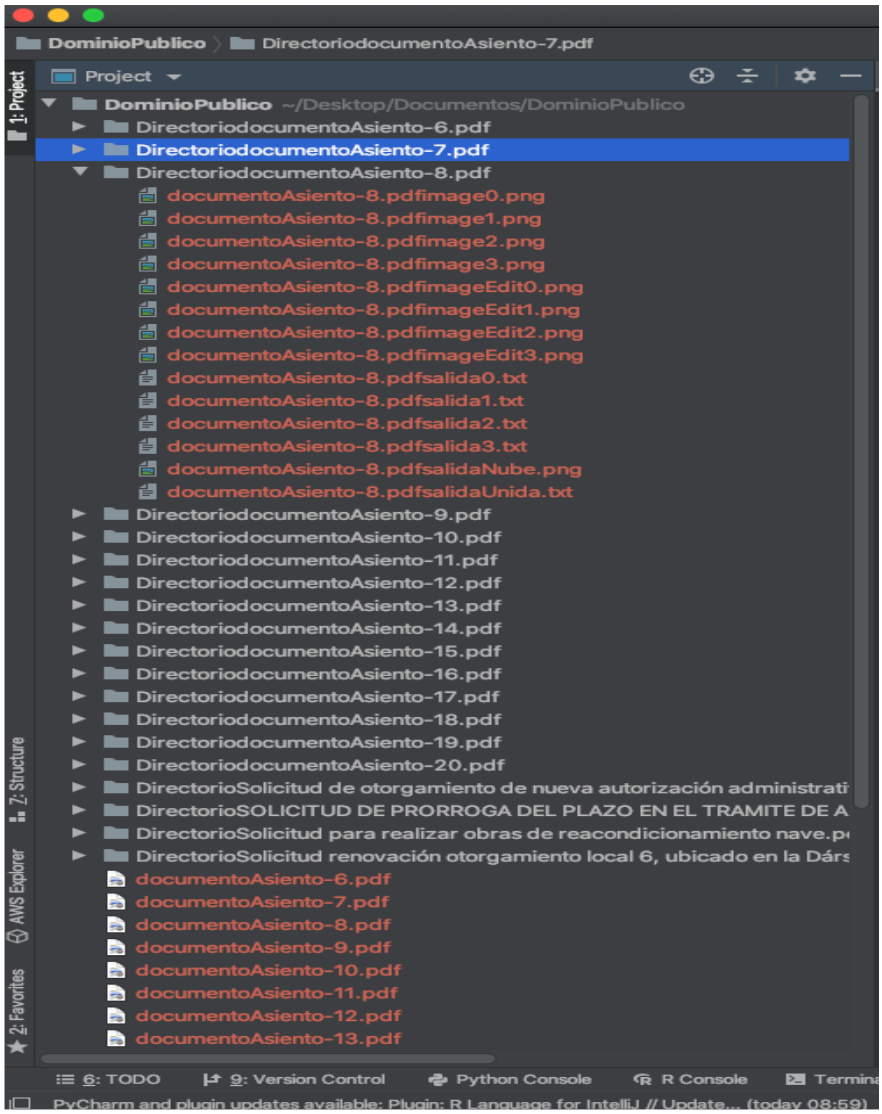


Figura 4.27. Trabajo con los Directorios

```
with open(proyect + "salida.csv", 'w', newline='') as csvfile:
    headers = ['Termino', 'TFid', 'Seccion']
    writer = csv.DictWriter(csvfile, fieldnames=headers)
    writer.writeheader()
    #writer = csv.writer(csvfile, delimiter=',')
    #writer.writerow(Headers)
    for row in range(0, collocations.shape[0]):
        writer.writerow({'Termino': str(collocations[row]), 'TFid': str(vector[row]),'Seccion': str('SISTINFO')})
#df = pd.DataFrame(vocabulario, columns=['Terminos', 'Tfid'])
#df.to_csv(proyect + 'salida.csv')
shutil.copy(proyect + 'salida.csv', nombredirectorio)
os.remove(proyect+ 'salida.csv')
print("TERMINOS CANDIDATOS A SER COLOCACIONES", collocations)
```

Figura 4.28. Implementación del etiquetado y conformado del csv

4.5 Filtros y clasificadores aplicados

Esta última parte de nuestro proyecto y que se corresponde al uso de *Weka* como herramienta de ML, se realiza con una herramienta de escritorio que ofrece *Weka* [65].

Como ya se ha expuesto anteriormente, de nuestros PDF hemos obtenido a través del uso de bibliotecas las palabras claves, así como su peso para poder trabajar de una manera más cómoda.

Obteniendo, así como salida de los PDF en primer lugar, unas imágenes de las que sacamos unos archivos de texto plano (*txt*) para posteriormente, tener las palabras con su peso representativo a partir de las veces que aparece en el texto y distintas fórmulas para obtener este peso.

Estas palabras claves se guardan en archivos *csv*, separadas por comas, realizando en este apartado del trabajo un etiquetado de cada una de estas palabras claves. Como se muestra en el código anterior ponemos de título (Términos, TFid, Sección), haciendo referencia al término, a su peso y a la etiqueta del departamento. La forma de etiquetación se basa en los distintos departamentos que componen la entidad: Dominio Público, Recursos Humanos, Secretaría General, Infraestructuras y por último Sistemas Información.

Por consiguiente, para poder así realizar el etiquetado hemos procedido ha sido escribir la etiqueta DOMINIO, RRHH, SECRETARIA, SISTINFO e INFRA siempre como última palabra del documento *csv*; así, de esta forma, posteriormente va a ser más fácil el procesamiento del lenguaje, simplemente accediendo a la última palabra o bien a la etiqueta determinada.

- *<recursos_humanos,5.248495242049359, RRHH>*

Esta muestra real hace referencia a un documento del departamento de Recursos Humanos y se especifica como tal con la etiqueta final empleada, para el entrenamiento de los modelos se guarda en un archivo *txt* tal como se ve en el ejemplo, pero sin estar "< >" incluidos.

Para la experimentación, previamente se han filtrado y modelado los datos, trasladándose a una cantidad de 29.312 términos a 3 archivos enriquecidos con formato *arff*. Estos archivos han sido creados de forma estándar mediante la consola de *Weka*, donde para su creación es necesario tener separadas en carpetas diferentes cada conjunto de frases con la misma clase; después se ha ejecutado la orden adecuada para automáticamente generar un documento *arff*, que contiene unas anotaciones para *Weka*.

Una vez preparamos el archivo *arff* se muestra como ha quedado abriéndolo como *txt* para mayor comodidad como se muestra en la Figura 4.29.

Debemos destacar la relación y los atributos y su declaración correspondiente, haciendo hincapié especialmente en el atributo clase que viene definido con los departamentos a los que hacemos referencia.

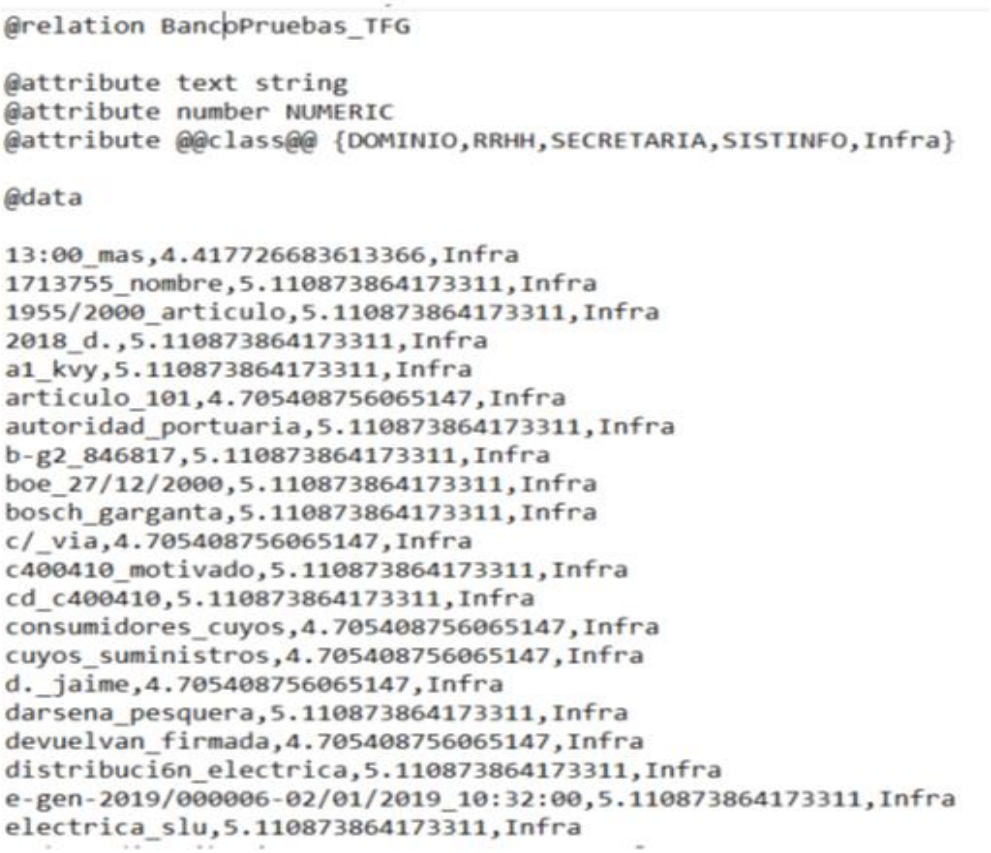
En nuestro caso, los términos que utilizamos para la clasificación proceden de los archivos CSV, al trasladarse a los archivos ARFF, quedan como se muestra en la Figura 4.28. Primero tenemos que definir como atributos *(@attribute*) los tipos de datos que vamos a utilizar.

El primer atributo *Text* del tipo *String* recopila todos los términos o palabras claves obtenidos.

El segundo atributo *number* del tipo *Numeric* representa el peso de cada término, así como su importancia en cada archivo.

Y por último el tercer atributo que es una clase, y que albergara los distintos departamentos donde se clasifican la totalidad de los textos.

Detalles para tener en cuenta: cuando migramos los datos desde el archivo CSV al archivo ARFF, surgen ciertos problemas debido a que en ciertos términos se introducen comas o comillas simples en medio de éstos provocando que tengamos que realizar la optimización de estos términos.



```
@relation BancoPruebas_TFG

@attribute text string
@attribute number NUMERIC
@attribute @@class@@ {DOMINIO,RRHH,SECRETARIA,SISTINFO,Infra}

@data

13:00_mas,4.417726683613366,Infra
1713755_nombre,5.110873864173311,Infra
1955/2000_articulo,5.110873864173311,Infra
2018_d.,5.110873864173311,Infra
a1_kvy,5.110873864173311,Infra
articulo_101,4.705408756065147,Infra
autoridad_portuaria,5.110873864173311,Infra
b-g2_846817,5.110873864173311,Infra
boe_27/12/2000,5.110873864173311,Infra
bosch_garganta,5.110873864173311,Infra
c/_via,4.705408756065147,Infra
c400410_motivado,5.110873864173311,Infra
cd_c400410,5.110873864173311,Infra
consumidores_cuyos,4.705408756065147,Infra
cuyos_suministros,4.705408756065147,Infra
d._jaime,4.705408756065147,Infra
darsena_pesquera,5.110873864173311,Infra
devuelvan_firmada,4.705408756065147,Infra
distribuci6n_electrica,5.110873864173311,Infra
e-gen-2019/000006-02/01/2019_10:32:00,5.110873864173311,Infra
electrica_slu,5.110873864173311,Infra
```

Figura 4.29. Archivo ARFF formado

De la misma manera podemos destacar que con las pruebas realizadas obtenemos casi 30.000 términos, concretamente 29.312 términos, de los que en torno al 1,25% presentan algún tipo de fallo.

Se realizan las correcciones oportunas y se procede a realizar la clasificación. Hay que mencionar también que *Weka* da problemas de la memoria ocupada en CPU si se toman archivos *ARFF* de un tamaño considerable, si bien es cierto que podemos aumentar la memoria que puede ocupar *Weka*, la solución más efectiva es el de trabajar con archivos más pequeños. De manera que en este proyecto se ha tomado la decisión de dividir nuestro banco de datos en 3 archivos *ARFF*.

En esta la Figura 4.30 se muestra la Interfaz *Weka*, donde se realiza la clasificación de los términos con los pesos de cada una de las clases. Así también se puede ver la cantidad de palabras o instancias que tenemos de cada sección o departamento.

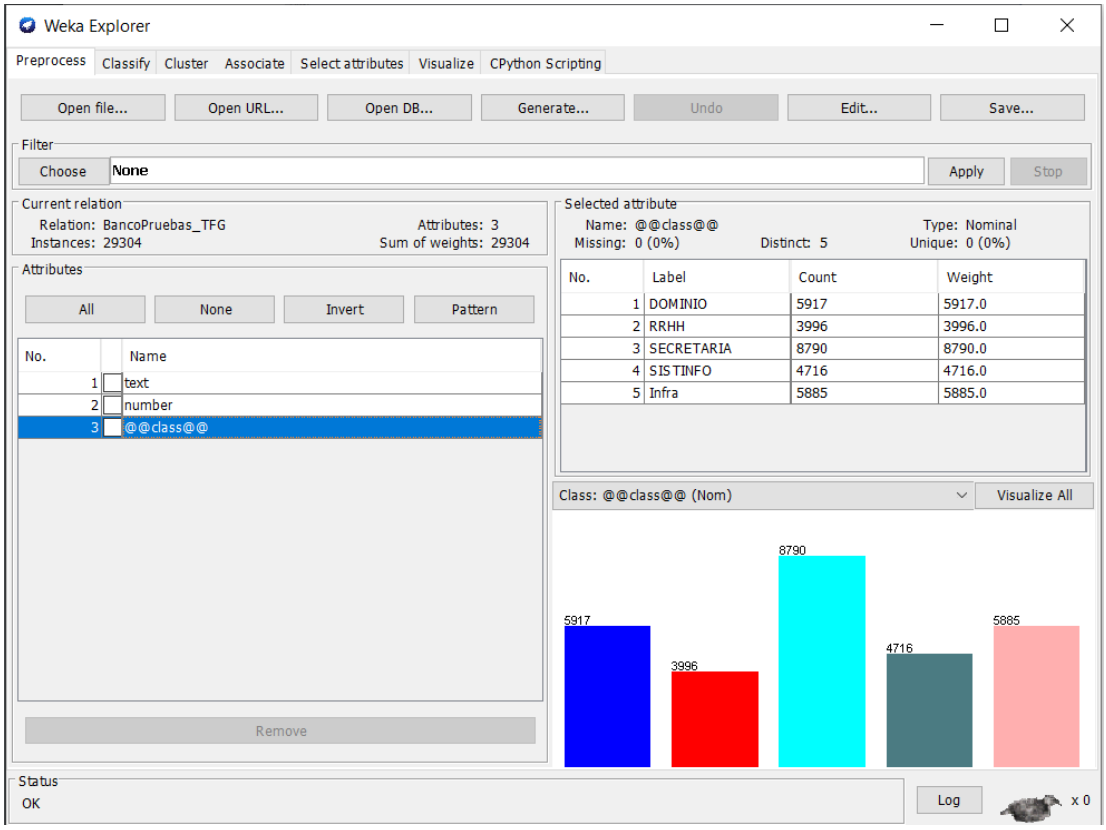


Figura 4.30. Weka implementación del ARFF

5 Resultados

En este capítulo se muestran los resultados obtenidos.

5.1 Introducción

En este apartado mostramos los resultados de la implementación de cada uno de los procesos que componen este TFG. Los tres primeros procesos se han obtenido tanto en *Windows* como en *MacOS*, mientras que el de *Weka* solo se ha verificado en *Windows* (Figura 5.1). Obteniendo así resultados favorables en todas las fases que componen esta memoria.

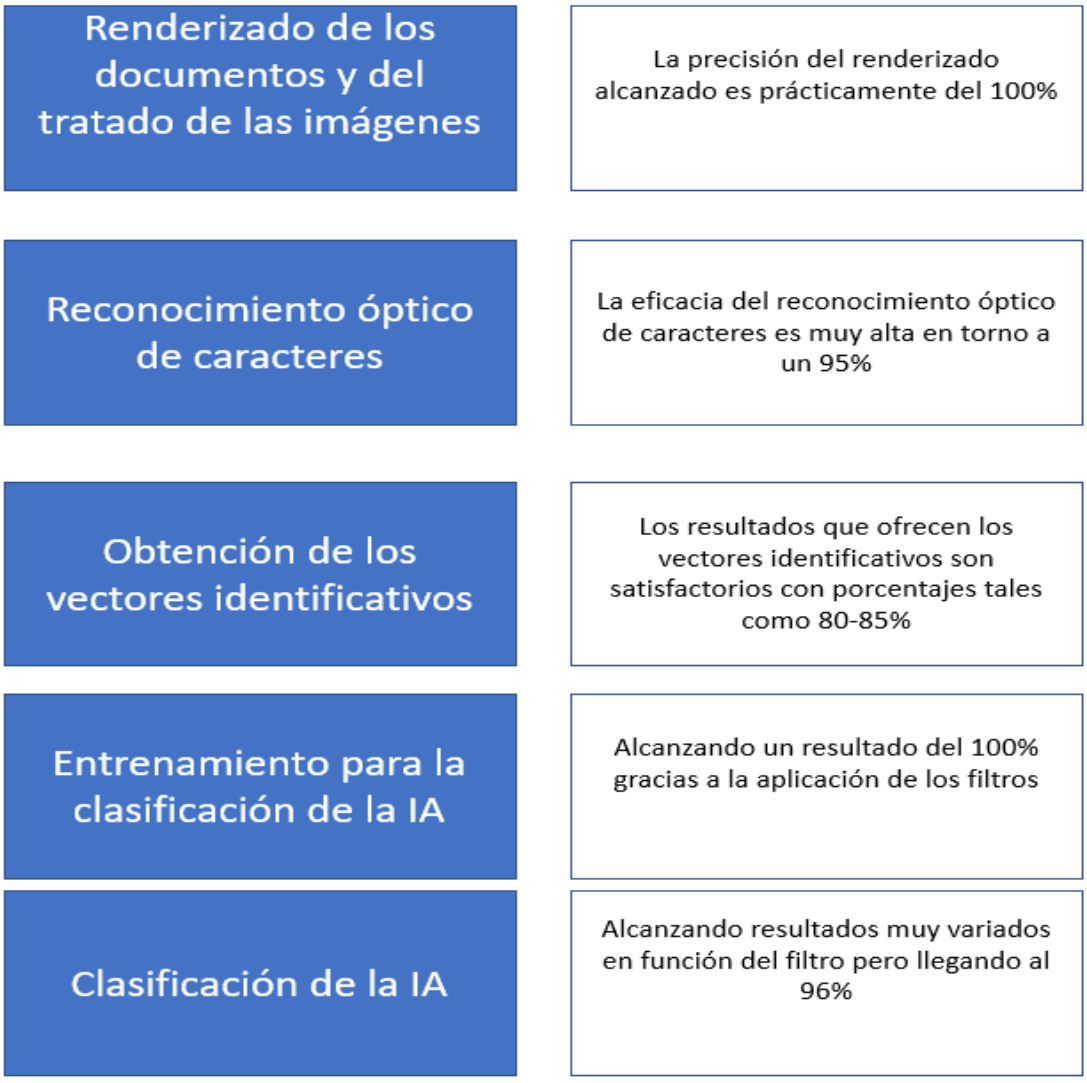


Figura 5.1. Resultados de todos los procesos que componen este TFG

5.2 Resultados del renderizado de los documentos y del tratado de las imágenes

Como mostramos en la Figura 5.2, nuestro código se encarga de dividir los PDF en imágenes, en función del número de páginas que tenga el PDF, creando así una imagen por página y posteriormente editando con *ImageMagick* el brillo y contraste y otros aspectos a mejorar de la imagen creando nuevas imágenes.

5.3 Resultados del reconocimiento óptico de caracteres

Una vez las imágenes han sido tratadas correctamente, se le aplica el OCR a cada imagen por separado y posteriormente se concatenan las salidas de texto plano en un solo documento obteniendo unos resultados satisfactorios, se pueden ver los resultados en la Figura 5.3. En el anexo D se muestra otros ejemplos del OCR y las imágenes.

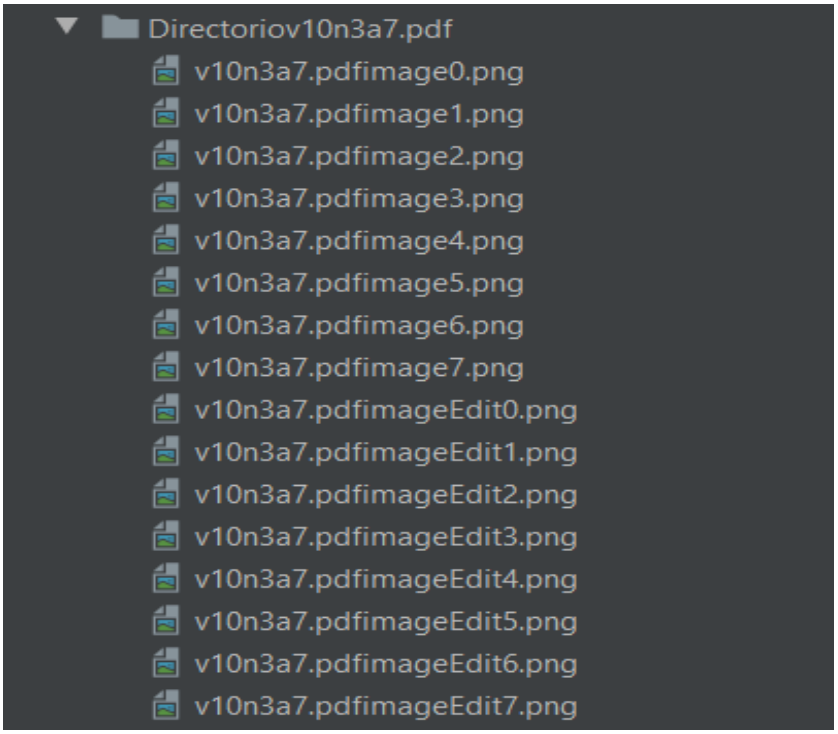


Figura 5.2. Tratado de las imágenes

5.4 Resultados de la obtención de los vectores identificativos

En cuanto a la obtención de los vectores identificadores, como se muestra en la Figura 5.4, la obtención de palabras claves a través del uso de la nube de palabras tiene éxito.

Si nos centramos en el uso de las *collocations*, con las que vamos a trabajar, se muestra que conseguimos con éxito quedarnos con las palabras más importantes del texto, en este caso los *ngramas* como se muestra en la Figura 5.5.

Al mismo tiempo, cabe destacar que antes del paso anterior se puede observar con cuantos documentos estamos trabajando, en el caso de esta prueba tenemos un documento y una longitud de tres páginas (Figura 5.6). A continuación, sacamos los *bigramas* (Figura 5.7).

Con los *bigramas* formamos los archivos *ARFF* que exportamos a la IA. Podemos ver que, si bien hay términos que podemos considerar descartables, la gran mayoría de ellos representan con mayor precisión el contenido del documento PDF.

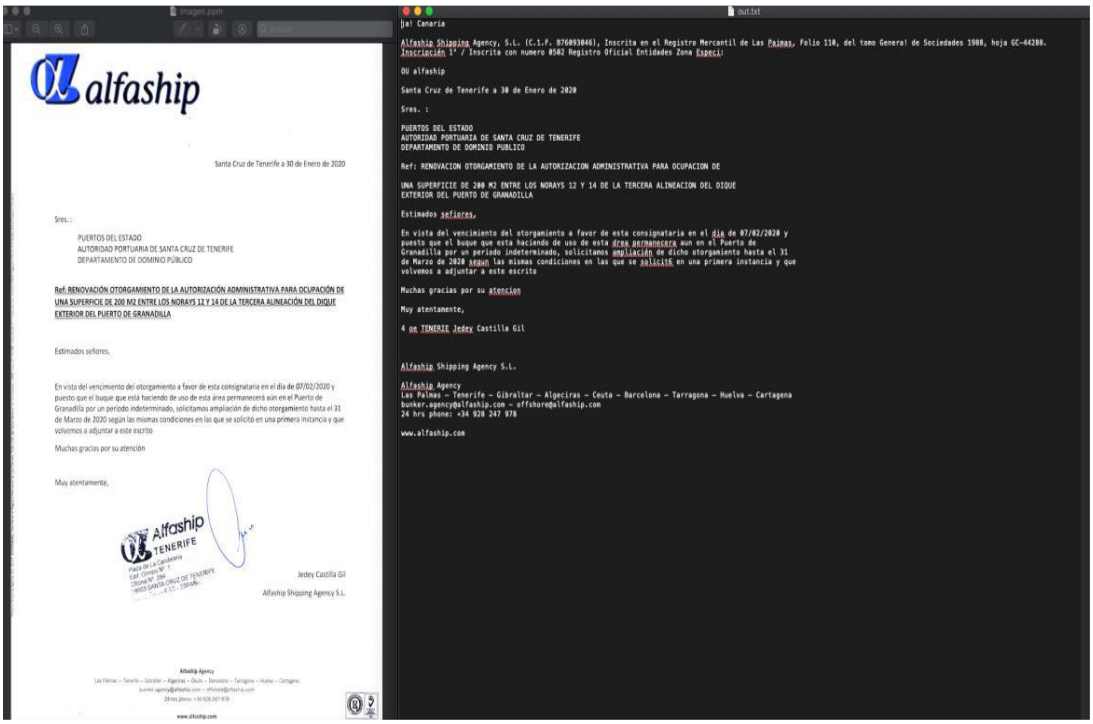


Figura 5.3. Resultado del OCR comparado con la imagen inicial

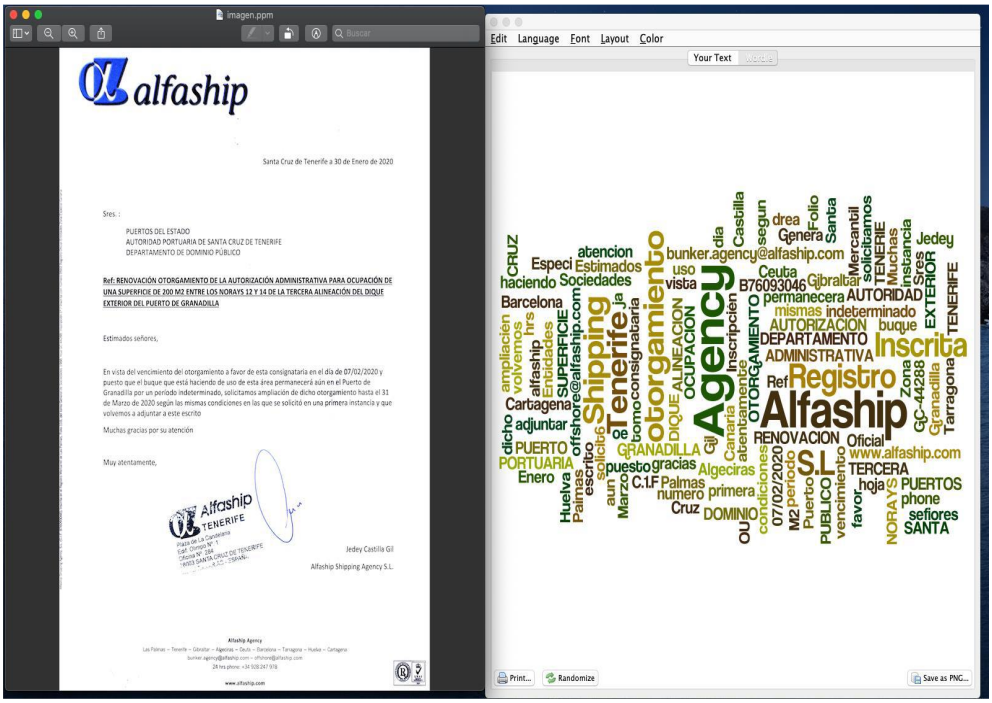


Figura 5.4. Resultados con la nube de palabras

```
['ja', '!', 'Canaria', 'Alfaship', 'Shipping', 'Agency', ',', 'S.L', '.', '(', '
C.1.F', '.', 'B76093046', ')', ',', 'Inscrita', 'en', 'el', 'Registro', 'Mercant
il', 'de', 'Las', 'Paimas', ',', 'Folio', '110', ',', 'del', 'tomo', 'Genera', '
!', 'de', 'Sociedades', '1988', ',', 'hoja', 'GC-44288', '.', 'Inscripción', '1°
', '/', 'Inscrita', 'con', 'numero', '0502', 'Registro', 'Oficial', 'Entidades',
 'Zona', 'Especi', ':', 'OU', 'alfaship', 'Santa', 'Cruz', 'de', 'Tenerife', 'a'
, '30', 'de', 'Enero', 'de', '2020', 'Sres', '.', ':', 'PUERTOS', 'DEL', 'ESTADO
', 'AUTORIDAD', 'PORTUARIA', 'DE', 'SANTA', 'CRUZ', 'DE', 'TENERIFE', 'DEPARTAME
NTO', 'DE', 'DOMINIO', 'PUBLICO', 'Ref', ':', 'RENOVACION', 'OTORGAMIENTO', 'DE'
, 'LA', 'AUTORIZACION', 'ADMINISTRATIVA', 'PARA', 'OCUPACION', 'DE', 'UNA', 'SUP
ERFICIE', 'DE', '200', 'M2', 'ENTRE', 'LOS', 'NORAYS', '12', 'Y', '14', 'DE', 'L
A', 'TERCERA', 'ALINEACION', 'DEL', 'DIQUE', 'EXTERIOR', 'DEL', 'PUERTO', 'DE',
'GRANADILLA', 'Estimados', 'sefiores', ',', 'En', 'vista', 'del', 'vencimiento',
 'del', 'otorgamiento', 'a', 'favor', 'de', 'esta', 'consignataria', 'en', 'el',
 'dia', 'de', '07/02/2020', 'y', 'puesto', 'que', 'el', 'buque', 'que', 'esta',
'haciendo', 'de', 'uso', 'de', 'esta', 'drea', 'permanecera', 'aun', 'en', 'el',
 'Puerto', 'de', 'Granadilla', 'por', 'un', 'periodo', 'indeterminado', ',', 'so
licitamos', 'ampliación', 'de', 'dicho', 'otorgamiento', 'hasta', 'el', '31', 'd
e', 'Marzo', 'de', '2020', 'segun', 'las', 'mismas', 'condiciones', 'en', 'las',
 'que', 'se', 'solicit6', 'en', 'una', 'primera', 'instancia', 'y', 'que', 'volv
emos', 'a', 'adjuntar', 'a', 'este', 'escrito', 'Muchas', 'gracias', 'por', 'su'
, 'atencion', 'Muy', 'atentamente', ',', '4', 'oe', 'TENERIE', 'Jedey', 'Castill
a', 'Gil', 'Alfaship', 'Shipping', 'Agency', 'S.L', '.', 'Alfaship', 'Agency', '
Las', 'Palmas', '-', 'Tenerife', '-', 'Gibraltar', '-', 'Algeciras', '-', 'Ceuta
', '-', 'Barcelona', '-', 'Tarragona', '-', 'Huelva', '-', 'Cartagena', 'bunker.
agency', '@', 'alfaship.com', '-', 'offshore', '@', 'alfaship.com', '24', 'hrs',
 'phone', ':', '+34', '928', '247', '978', 'www.alfaship.com']
emartrus@iMac-de-admin-2 prueba %
```

Figura 5.5. Resultado con los ngramas

```
['SOLICITUD DE MEDIDAS CORRECTORAS DE SITUACIONES A CORREGIR POR LA AUTORIDAD PORTUARIA.pdf']
LISTADO FINALIZADO
longitud de la lista =  1
SOLICITUD DE MEDIDAS CORRECTORAS DE SITUACIONES A CORREGIR POR LA AUTORIDAD PORTUARIA.pdf
[<pil.PpmImagePlugin.PpmImageFile image mode=RGB size=1654x2336 at 0x1ADA6395C18>, <pil.PpmImagePlugin.PpmImageFile image mode=RGB size=1654x2336 at 0x1ADA6395978>, <pil.PpmImagePlugin.PpmImageFile image mode=RGB
size=1654x2336 at 0x1ADA6395A90>]
longitud de paginas:  3
SOLICITUD DE MEDIDAS CORRECTORAS DE SITUACIONES A CORREGIR POR LA AUTORIDAD PORTUARIA.pdfimage0.png
SOLICITUD DE MEDIDAS CORRECTORAS DE SITUACIONES A CORREGIR POR LA AUTORIDAD PORTUARIA.pdfimage1.png
SOLICITUD DE MEDIDAS CORRECTORAS DE SITUACIONES A CORREGIR POR LA AUTORIDAD PORTUARIA.pdfimage2.png
```

Figura 5.6. pasos previos en la obtención de los vectores identificativos


```
COLLOCATIONS ['07_8420', '0730_u03', '11g_sesosety', '12158_8qap|', '13_si0e-u-sz', '13:45:00o10z-86-20_loveiay', '1809_opfhsweioui', '19p_odionde', '1a0dec9c0d_17d40f1c644', '1a7857de7bd7_98dfba9f31', '1ebb4623a_
observaciones', '1ep_up0819|', '1eua9_ond', '2020_querido', '2102_wauevedo', '2g_openiedy', '2m_cosmno', '31_2b', '313.570_validez', '34483nfl_bc', '34r13nal_3g', '3cdaf4d8b396_1a7857de7bd7', '3g_zno', '402_sqls¥'
, '415_daa', '49_38001', '51_bue', '5501_presidencia', '59159014_1ebb4623a', '754ad6b9e0ce79a1728c270ea9fbb6dfcedfct_59159014', '7f39a6e95fac7725b_1a0dec9c0d', '80-62_tousiay', '8qap|_9402', '9102_020,80', '98dfba
9f31_754ad6b9e0ce79a1728c270ea9fbb6dfcedfct', 'a.qweinid_of', 'aane_96:00', 'adjuntos_nombre', 'administracionespublicas.gob.es/valida_código', 'ae_oueuipioeme', 'amo_wiholion', 'anata_fant', 'and|_wuanueioie', 'a
poved|_0608', 'apsep_odusad', 'aron_seseidue', 'arr_eloz', 'art_31', 'b99rg_eunoyur', 'baam=eummaoauayyamamaeaeaaeaeaaeaaeaeaeeeeeeeeseee_eee', 'bc_znyd', 'bred_soperynn', 'brevedad_posible', 'bue_wuaioa', 'buepio
s_sa', 'buupurh_sers', 'can_ex', 'carlos_enrique', 'comunes_justificante', 'comunicarlo_motivadamente', 'congnd_oanuop', 'cordial_saludo', 'cosmno_servicics', 'código_seguro', 'd_ent', 'd._carlos', 'daa_sopenosy',
 'debe_comunicarlo', 'debera_modificar', 'dei_cémputo', 'dias_habiles', 'documentaci6on_adjunta', 'documentacion_fisica', 'e/uwai_2g', 'e2yusni_a6', 'e7|_6102-60-20', 'eaay_buupurh', 'eauraag_eaay', 'efectos_dei',
 'eid_92', 'ejqivodin_aseq', 'electronicas_anexadas', 'eloz_euopeny', 'enrique_gonzalez', 'entendera_realizada', 'envies_cumplimentado', 'epeweidu_wee', 'esiepa2v02_ueaap', 'espana_tel', 'establezcdis_nuevas', 'eu
abe_uuey', 'euopeny_rated', 'eyr04_urtsetorert', 'f_oyuounauuns', 'fant_key', 'fisico_ongen', 'geiser_geiser-b5c0-a377-3ed9-4628-8a74-e42b-4bc2-c9f5', 'geiser-0700-0b7f-6692-44e9-b03f-daa9-a564-a682_hash', 'gonzal
ez_pérez', 'gpze_euabe', 'gvainoiny_adv', 'gwar_b99rg', 'id_onto', 'imagen_fiel', 'imagenes_electronicas', 'innabil_ambito-prefijo', 'is_souses', 'jouiuad_vorrewewid99', 'key_opuens', 'li0z_is6e9', 'mayor_brevedad
', 'me08170720200110141029.pdf_tamanio', 'nabil_siguiente', 'naciones_28042', 'nade_f', 'nombre_me08170720200110141029.pdf', 'norma_permita', 'normativa_vigente', 'nv_id', 'n¥id_oonnois', 'o8:1u09_44', 'odn_1ep',
'odusseg_nowow', 'oh_oi', 'oise6_13', 'ond_wued', 'onto_is', 'oonnois_soseibul', 'op.ense_oq', 'opsance_oup', 'opumasuen_iaqey', 'origen/centro_secretaria', 'ouepusre|_sepipew', 'oueuipioeme_seppese2', 'ourpuare|_
sepypeu', 'ovehiuod_opelode', 'owoa|_sejecypn', 'oy_opo', 'pa_ueuioce', 'partenon,10_campo', 'pasado_mes', 'pedaue_e7|', 'permita_expresamente', 'piazo_fyado', 'poun_va', 'proximo_viernes', 'puede_verificar', 'rat
ed_loyegy', 'recibido_escrito', 'resumen/asunto_segundo', 'risody|_7', 'rpoupny_34483nfl', 's2uoreasosao|_ourpuare|', 'sa_souisiw', 'salida_resumen/asunto', 'se103981_soid:2und', 'seiopelege_ant', 'seousyi0326189_
se|', 'sepencepe_epeweidu', 'sepeprsesou|_58', 'sepiden_s|', 'sepipou_urpode', 'seppese2_wo', 'servicics_can', 'sesosety_zeuke', 'seucreuop_savoiouangns', 'siguiente_salvo', 'sivewedigdp_12158', 'sod_ep29', 'sodpo
rved_soise6', 'soisendoud|_wore', 'soje-sadsa_seousyi0326189', 'soperynn_somneuirred', 'sopevequ02_seiopelege', 'soseiful_2102', 'sr._d.', 'su02_80-62', 'sued_whew', 'susiee_4g8c', 'syaued_50', 's|_levosied', 'tra
mitacion_origen/centro', 'tun_9102', 'u03_op.ense', 'udewawnied_soueiryev0g', 'ueaap_savoruengns', 'ued_pa', 'ueqavad_anb', 'ui_rpoupny', 'umudy_0ge', 'uodwy_aron', 'uoiadiu2e0q_ourpuare', 'uoiaeypeice_14', 'uoie_
90', 'uoinduseg|_uote', 'uoizdusseq|_a4', 'uoseywen_jouiuad', 'upioenyis_e1', 'uprodis2saq|_11891209', 'usb_vew', 'v8_405', 'va_uge', 'validacion_vi', 'validez_copia', 'veo_oye1qu09', 'verificacion_geiser-b5c0-a37
7-3ed9-4628-8a74-e42b-4bce2-c9f5', 'very_34r13nal', 'vi_d', 'vodka_uodwy', 'vole_wwaigo', 'voo_uodego', 'wallyb_amo', 'wetdtsetost_sepipaw', 'wiholion_be', 'wihwnlnod_gvainoiny', 'wo_opsance', 'wore_saip42e9', 'ws
```


Figura 5.7. Resultados de los bigramas

5.5 Resultados del entrenamiento para la clasificación

Una vez tenemos los archivos, los importamos dentro de la aplicación de escritorio de *Weka*. A continuación, a los términos les aplicamos un filtro que convierte las *String* a vectores de palabras para su correcta manipulación. Sin este filtro, las bibliotecas de *Weka* no serían capaces de leer los datos.

El siguiente paso después del filtrado es la instanciación del modelo a entrenar creando un modelo vacío y sin datos.

Se ajustan las diferentes opciones disponibles para dicho modelo mediante la técnica de validación cruzada.

Entrenando el modelo creado, finalmente se estima su nivel de eficiencia mediante el nivel de aciertos y fallos y los diferentes tipos de errores, también con la *precisión* y *recall* posteriormente explicados.

La técnica de *validación cruzada* (o *cross validation*) es una manera de entrenar los modelos, de forma que los datos obtenidos se utilizan al máximo.

La forma normal del entrenamiento consiste en tener dos conjuntos de datos de diferente tamaño y con el de mayor tamaño (por ejemplo, un 80% de los datos totales), utilizarlos para entrenar el modelo y los restantes (el 20%), para comprobar su resultado óptimo.

Mediante *cross validation*, los datos totales son particionados también en dos subconjuntos, pero esta vez se toman N particiones del total de datos, usando N-1 para entrenar y 1 para la clasificación en departamentos.

La diferencia es que este proceso se repite varias veces cambiando las particiones escogidas consiguiendo con un solo conjunto de datos la simulación de tener varios conjuntos.

Específicamente, en este estudio se ha utilizado la validación cruzada de 10 vueltas. Esta opción es la que viene por defecto en la biblioteca de Weka y se ha seguido el consejo, puesto que en la mayoría de las veces que se ha experimentado los parámetros estándar suelen ser óptimos.

5.6 Resultados de la clasificación

A continuación, mostramos como construimos el clasificador y los distintos clasificadores:

1. *ZeroR*: en primer lugar, seleccionaos *ZeroR* (arriba a la izquierda). Este clasificador clasifica a todos los datos con la clase de la clase mayoritaria. Es decir, si el 90% de los datos son positivos y el 10% son negativos, clasificará a todos los datos como positivos. Es conveniente utilizar primero este clasificador, porque el porcentaje de aciertos que obtenemos con él es el

que tienen que superar con el resto de los clasificadores. Antes de lanzarlo, seleccionamos la opción *Crossvalidation* para hacer el test. Y pulsamos el botón Start.

Como se muestra en la Figura 5.8. El resultado de clasificación que tenemos que batir es de un *41.3456 %* de acierto, en la clasificación de los departamentos.

Obteniendo un 41% de aciertos (lógico, puesto que hay diferencia de cantidad de palabras entre las cinco clases). En el porcentaje de aciertos desglosado por clase (*TP rate*, o *True Positive Rate*) vemos que en la tercera clase la acierta al 100% (*Secretaria, TP rate* = 1) y las otras dos las falla (*TP rate* = 0%).

Dada que la manera en la que funciona *ZeroR*: sólo acierta la clase mayoritaria, o la primera, en caso de que ninguna sea mayoritaria. En este caso interpreta que *Secretaria* es la mayoritaria por su peso como se muestra en la Figura 5.9.

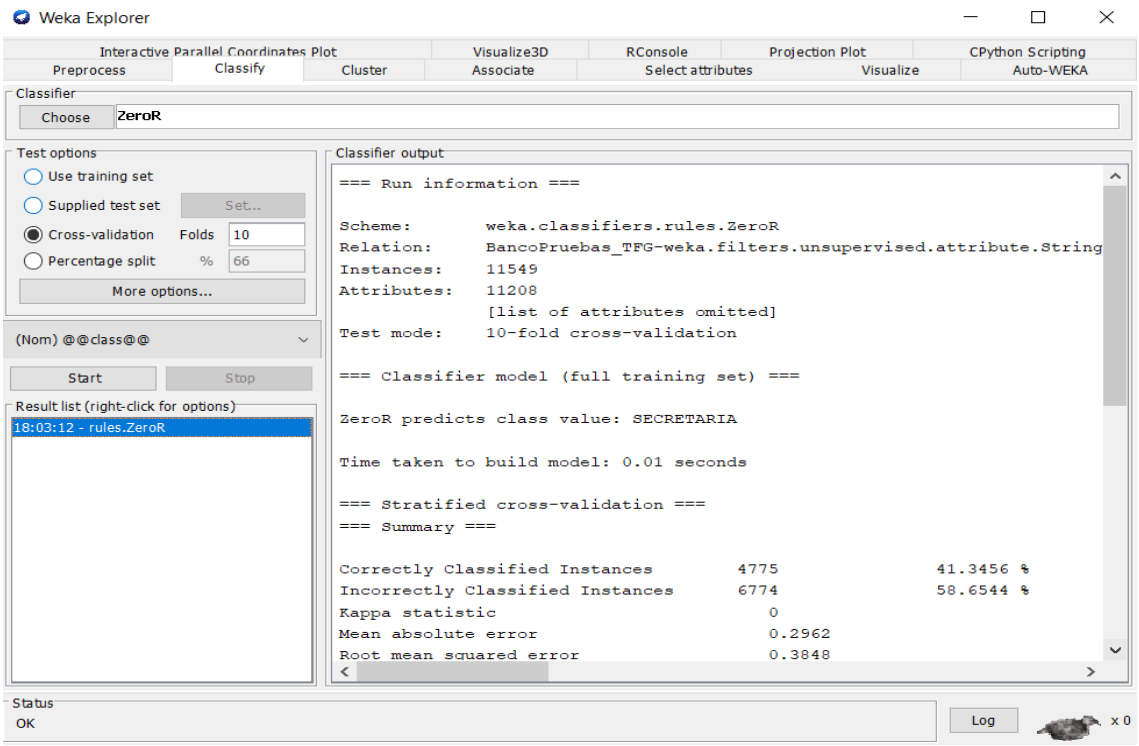


Figura 5.8. Clasificador ZeroR

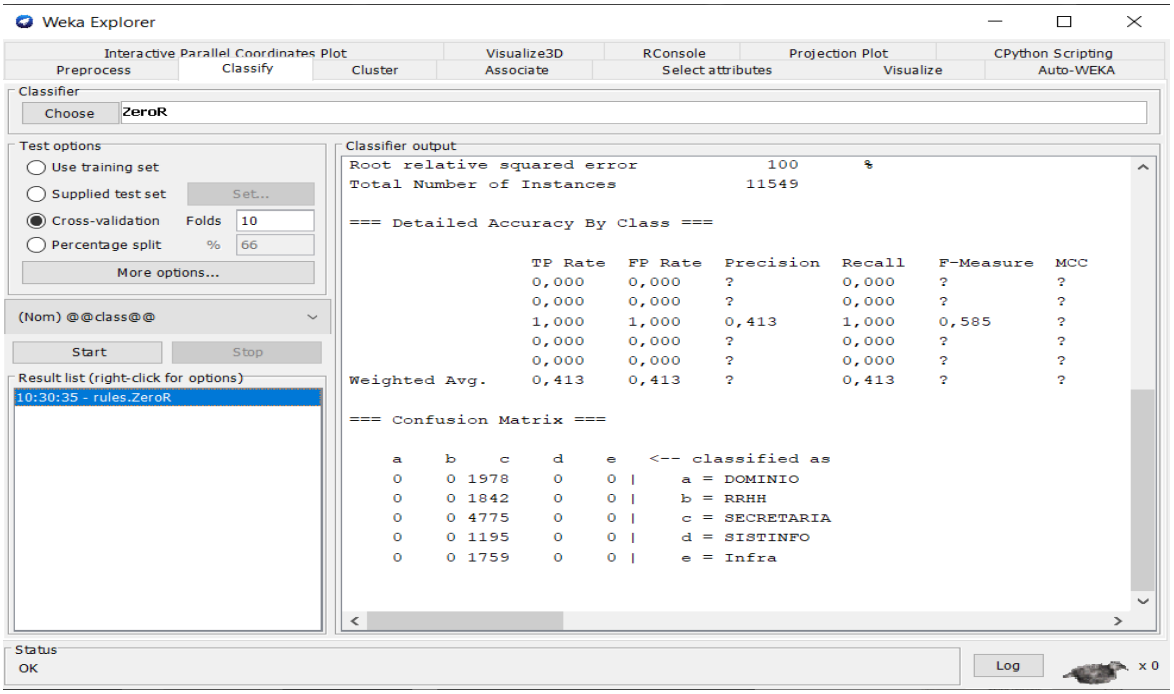


Figura 5.9. Resultado de la clasificación con ZeroR

Ya sabemos que el porcentaje de aciertos a superar es el del *41%*. Vamos a probar ahora con otro clasificador. Por ejemplo, el clasificador *PART*, que construye reglas y que mostramos sus resultados en la Figura 5.10.

2. PART: siguiendo con los clasificadores probamos PART, eligiéndolo dentro de los distintos clasificadores disponibles y pulsamos Start.

   Una vez se construye el modelo y se realiza el entrenamiento se muestra que ahora el porcentaje de aciertos es de un *96%*, muy superior al del caso base (*ZeroR*). En el desglose de los aciertos por clase como se puede ver en la Figura 5.10, vemos que todas se aciertan bastante bien, siendo *Secretaria* la que tiene más aciertos (*99,7%*) e *Infraestructuras* la peor (*83%* de aciertos).

   Ofreciendo un resultado bastante equilibrado en los aciertos de las distintas clases, lo que tiene sentido, puesto que partíamos de un conjunto de datos balanceado (de este modo es posible que la clase mayoritaria tuviera mayor porcentaje de acierto que las minoritarias).

A continuación, probamos a seleccionar y ejecutar otros algoritmos y ver los resultados. En la siguiente gráfica se puede ver el *J48* (*tree*, árbol de decisión).

3. *J48*: como se muestra en la Figura 5.11, continuamos nuestras pruebas con el modelo *J48*, este modelo también tiene un porcentaje muy alto y similar a los anteriores con un *96,32%* de acierto en la clasificación.
4. *Redes Bayesianas o Bayes Net*: una red *bayesiana* depende de las ecuaciones de *Bayes*. Una red bayesiana de este tipo es representada con un grafo acíclico dirigido, cuyos nodos representan variables y las aristas las relaciones condicionales; cada nodo tiene asociada una función de probabilidad dependiente de los nodos que lo apuntan. Cada variable es representada de forma booleana con sus probabilidades de que el evento pueda ocurrir o no. En la Figura 5.12 se muestra un ejemplo bien formado de redes bayesianas.

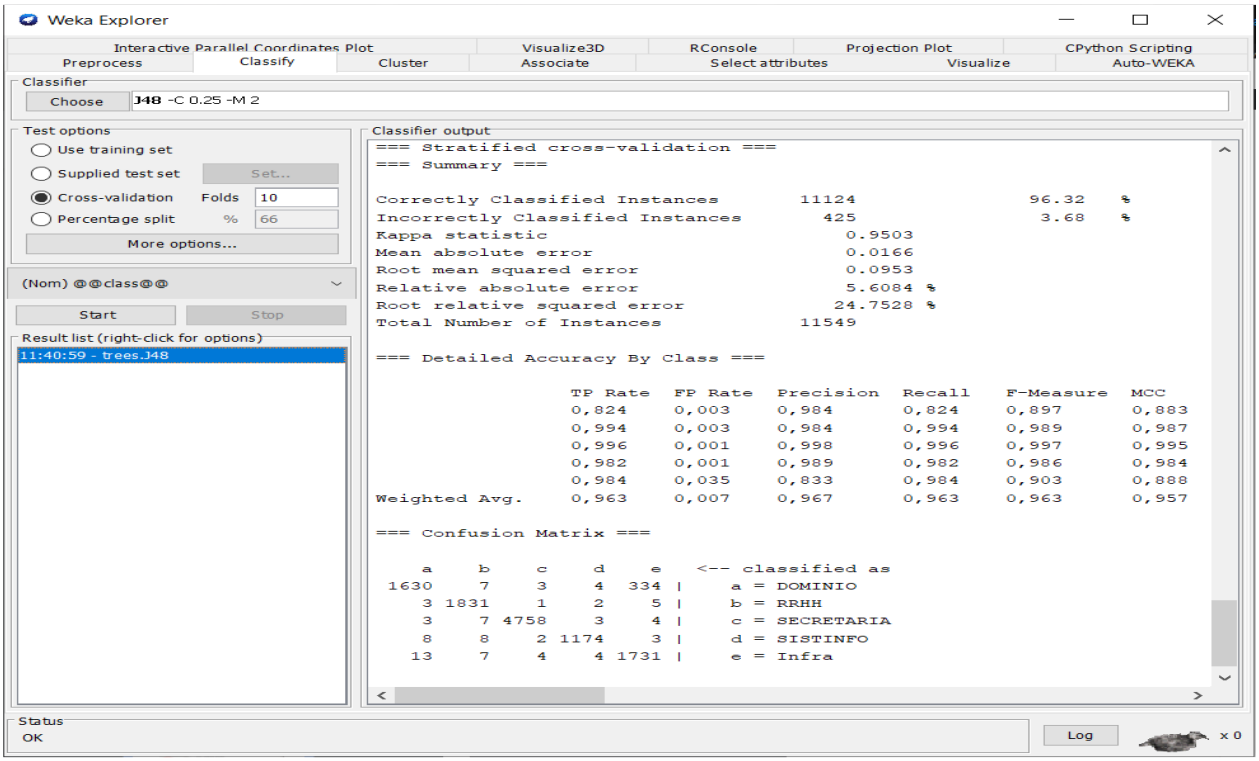


Figura 5.10. Clasificación con J48

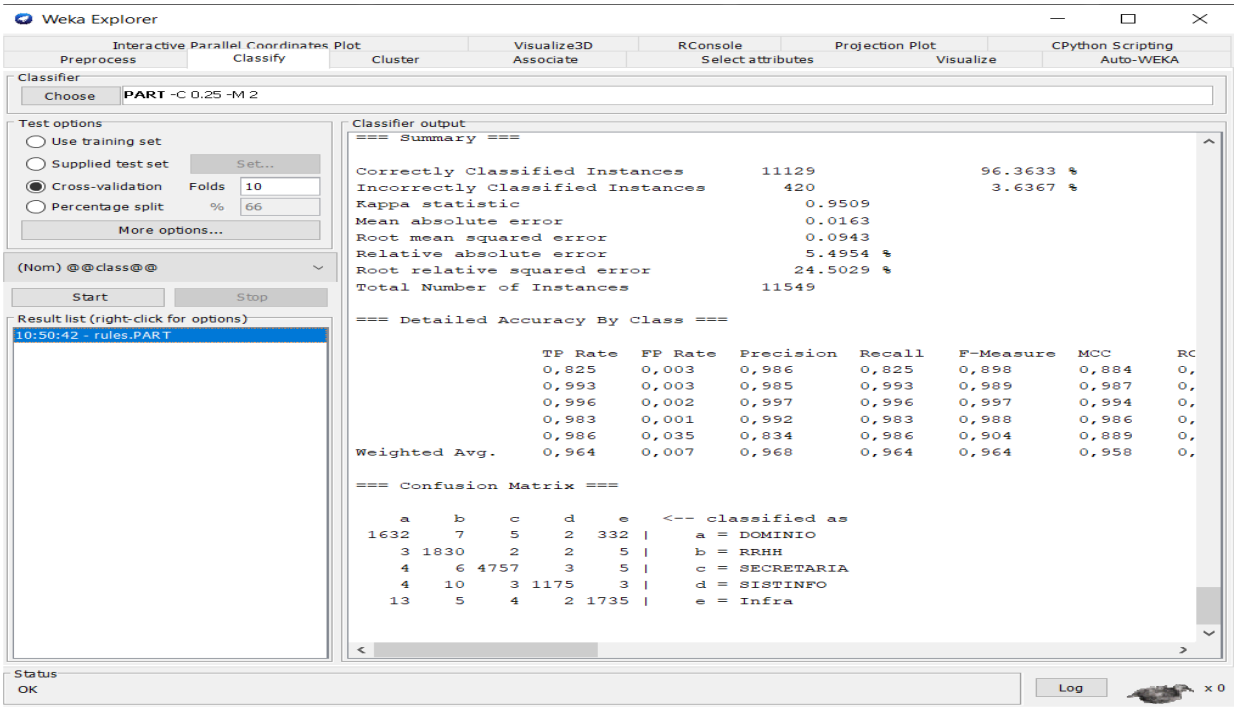


Figura 5.11. Resultado de la clasificación con PART

5. *SMO*: El modelo *SMO* que es un algoritmo que utiliza Weka para implementación de *Support Vector Machine* (*SVM*), sin embargo, ha tenido unos resultados muy inferiores a los anteriores. A pesar de la naturaleza de los diferentes *kernels* que se han utilizado, la mayoría de los resultados obtenidos han sido proporcionales entre ellos e igual de buenos en la mayoría de los casos. Los resultados del modelo *SMO* se muestran en la Figura 5.12.

Finalmente, se encuentra la red neuronal, que, a pesar de las esperanzas depositadas en este modelo, tras su correcto entrenamiento con sus diferentes opciones de nodos y capas ocultas, al llevar un tiempo con su ejecución; el proceso lanza un error por falta de memoria en todos y cada uno de los casos probados.

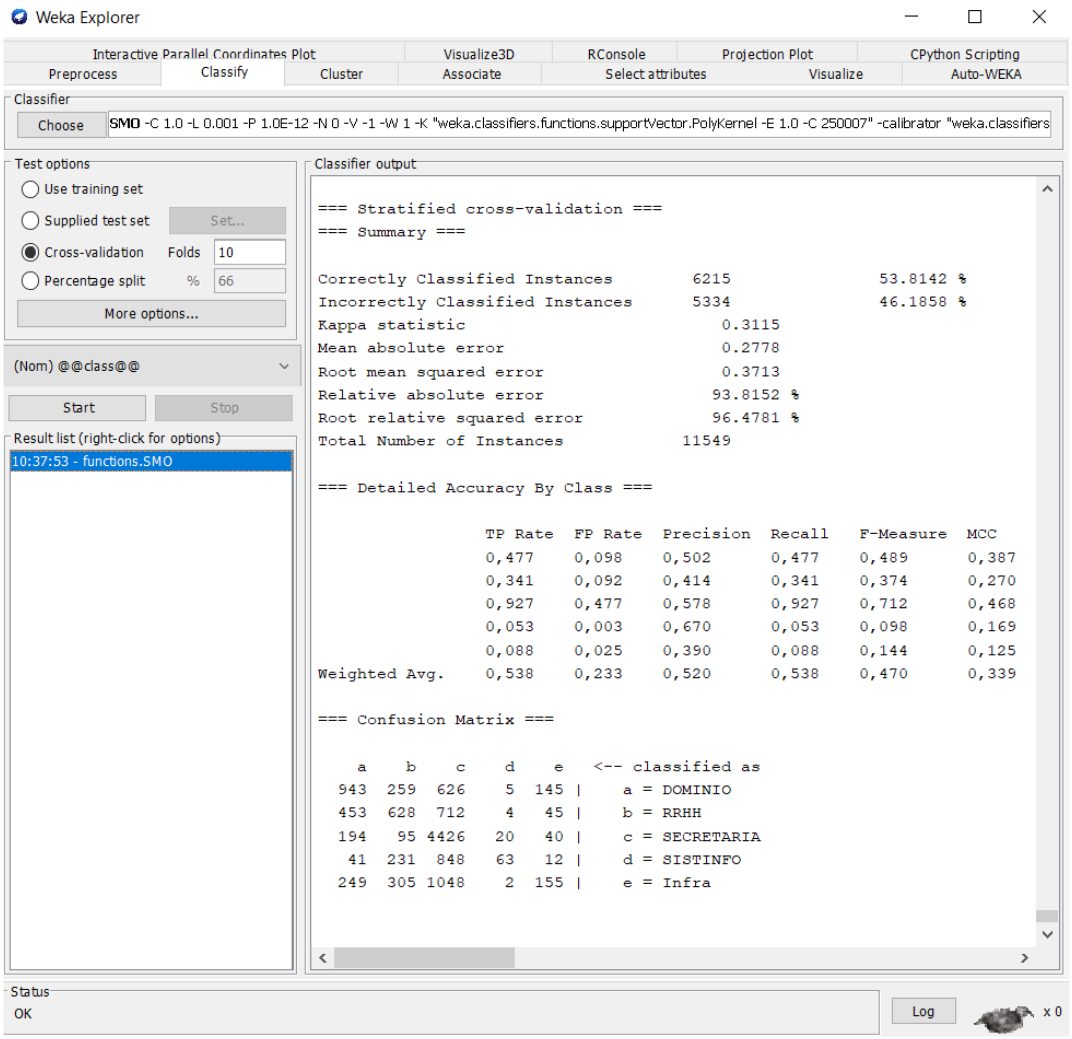


Figura 5.12. Modelo SMO

6 Conclusiones y posibles ampliaciones

En este capítulo se muestran las conclusiones adquiridas tras realizar este proyecto, además de la presentación de una serie de posibles ampliaciones que se podrían llevar a cabo en el TFG.

6.1 Conclusiones

Gracias al avance de la tecnología y la aparición de la IA y su implementación en la vida cotidiana de todos, se ha facilitado enormemente la realización de tareas diarias. Así mismo, en la actualidad se habla cada vez con más frecuencia del uso de la IA y las ventajas y facilidades que puede aportar. Sin embargo, esta tecnología está aún en desarrollo y lejos de llegar a conocer su enorme potencial.

En este TFG, se realiza un estudio con las distintas herramientas de ML y que pueden cubrir necesidades que repercuten a las distintas entidades u organismos pertenecientes al Estado. Facilitando así numerosas tareas y reduciendo así una gran cantidad de horas de trabajo.

En el estudio se descubren las distintas herramientas y se decide trabajar con Weka por su sencillez, compatibilidad con base de datos y su flexibilidad en la implementación con Java y otros lenguajes como Python.

Del mismo modo, en este proyecto también se ha llevado a cabo un estudio sobre la posibilidad de trabajar con documentos PDF, pertenecientes a las entidades mencionadas anteriormente y que pueden servir como fuente de datos para la IA. Por consiguiente, se realiza un estudio sobre las distintas herramientas que podemos emplear de OCR para poder trabajar con el contenido de los documentos y con ello eliminar los posibles defectos de los PDF al tratarlos como imágenes, tales como: marcas de aguas, firmas, sellos, escritos a mano... Usando bibliotecas como ImageMagick y Tesseract, para el tratado de las imágenes y el OCR correspondientemente. La conclusión obtenida ofrece un resultado muy favorable a estas técnicas.

Usando nubes de palabras y eliminando los términos menos significativos o más comunes del texto, se completa la posibilidad de obtener las palabras más importantes y que describan con mayor precisión cada documento. Se tiene como conclusión que es posible obtener una gran fuente de datos que sean fieles a los documentos recibidos con gran resultado.

Por otro lado, una vez se tienen ese gran banco de datos, se conforman los archivos CSV en primer lugar, y a continuación el archivo ARFF para poder realizar el entrenamiento y la validación de los datos con la IA. Este entrenamiento y validación se realiza en Weka, usando los distintos modelos clasificadores y comprobando los resultados obtenidos.

Finalmente, hay que destacar que durante el desarrollo de este TFG se ha puesto en práctica gran parte de los conocimientos aprendidos durante el grado, tales como desarrollar código en un lenguaje de programación desconocido e implementación de bibliotecas, el tratamiento de las imágenes para el renderizado de los PDF, el trabajo con la IA y la conformación de los datos, muy similar al de una base de datos. Y, además, se han puesto en práctica distintas habilidades de ingeniería para realizar búsquedas, interpretación y puesta en práctica de la información obtenida en las diferentes áreas trabajadas y estar capacitado para la realización y evaluación de pruebas. Gracias al desarrollo de este TFG, se ha obtenido un amplio conocimiento de las tecnologías utilizadas desconocidas con anterioridad.

6.2 Posibles ampliaciones

En este apartado se especifican algunas posibles ampliaciones a este TFG que pueden llevarse a cabo en futuras extensiones:

- *Clasificar documentos con otro fin*: uno de los objetivos de este proyecto ha sido lograr que se pudieran clasificar documentos haciendo uso de la IA, por lo que podríamos considerar implementar una clasificación en función de las emociones, de si el lenguaje que describe el documento es malsonante, es sexista o no… Sin embargo, tal como se ha indicado en la realización del TFG no se han dispuesto estas variaciones, ya que, en cuanto a las emociones, habría que describir qué tipo de términos indican y crear un diccionario con ellas al igual que con los otros casos, determinando que tipo de términos corresponderían a que las palabras utilizadas sean o no sean sexistas o malsonantes.
- *Incluir control desde el móvil*: actualmente las personas realizan sus tareas desde el dispositivo móvil, es por esto, por lo que una ampliación interesante sería

permitir que la clasificación de documentos PDF se pudiera realizar desde el móvil, o al menos poder mandar las palabras claves de los documentos a una base de datos.

- *Implementar distintas herramientas de ML*: como ya dijimos nosotros implementamos Weka pero en los anexos se podría ver como se probó Google Cloud e IBM Watson y se explica porque se descartaron por problemas con su implementación y por pagos.

Glosario

OCR	Optical Character Recognition
API	Application Programming Interface
IA	Inteligencia Artificial
ASCII	American Standard Code for Information Interchange
SVHN	The Street View House Numbers
SVT	The Street View Text Dataset
IBM	International Business Machines Corporation
OMS	Organización Mundial de la Salud
RPA	Robotic Process Automation
PDF	Portable Document Format
RRHH	Recursos Humanos
OOP	Orientación a Objetos
SEO	Search Engine Optimization
PNG	Portable Network Graphics
RGB	Red, Green, Blue
NLTK	Natural Language Toolkit
PLN	Procesamiento de Lenguaje Natural
JDBC	Java Database Connectivity
SQL	Structured Query Language
CSV	Comma-Separated Values
ARFF	Attribute-Relation File Format
TFG	Trabajo de Fin de Grado

Referencias

[1] “What is Scanner?“, TechTarget.com, 2021. [Online]. Available: https://whatis.techtarget.com/ definition/scanner. [Accessed: 28- Sept- 2021].

[2] “What is artificial intelligence?", TechTarget.com, 2021. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence. [Accessed: 28- Sept- 2021].

[3] "Intelligent Character Recognition (ICR)“, TechTarget.com, 2021. [Online]. Available: https://whatis.techtarget.com/definition/intelligent-character-recognition-ICR. [Accessed: 28- Sept- 2021].

[4] “Portable Document Format (PDF)“, TechTarget.com, 2021. [Online]. Available: https://whatis.techtarget.com/definition/Portable-Document-Format-PDF. [Accessed: 20- Sept- 2021].

[5] “What is Soft Copy?", TechTarget.com, 2021. [Online]. Available: https://whatis.techtarget.com/definition/soft-copy. [Accessed: 19- Sept- 2021].

[6] “What is Word Processor?", TechTarget.com, 2021. [Online]. Available: https://searchwindowsserver.techtarget.com/definition/word-processor. [Accessed: 10- Sept- 2021].

[7] “What is bit Map?", TechTarget.com, 2021. [Online]. Available: https://whatis.techtarget.com/definition/bit-map. [Accessed: 14- Sept- 2021].

[8] “What is pattern Recognition?", TechTarget.com, 2021. [Online]. Available: https://whatis.techtarget.com/definition/pattern-recognition. [Accessed: 05- Sept- 2021].[9] “OCR para caracteres impresos basados en Árboles Binarios? ", 2021. [Online]. Available:

http://grupo.us.es/gtocoma/pid/pid10/OCRarbolbinario.htm. [Accessed: 01- Sept- 2021].

[10] "The Street View House Numbers (SVHN) Dataset", 2021. [Online]. Available: http://ufldl.stanford.edu/housenumbers/. [Accessed: 01- Sept- 2021].

[11] Repository by @opencars / alpr, 2021. [Online]. Available: https://github.com/openalpr/openalpr. [Accessed: 01- Sept- 2021].

[12] GitHub - qjadud1994/CRNN-Keras: CRNN (CNN+RNN) for OCR using Keras / License Plate Recognition. (n.d.). Retrieved October 1, 2021, from https://github.com/qjadud1994/CRNN-Keras.

[13] Number plate detection with Supervisely and Tensorflow (Part 1) | by Supervise.ly | Towards Data Science. (n.d.). Retrieved September 30, 2021, from https://towardsdatascience.com/number-plate-detection-with-supervisely-and-tensorflow-part-1-e84c74d4382c.

[14] VPAR SERVER – Analíticas de Tráfico y Control de Accesos. (n.d.). Retrieved September 30, 2021, from https://www.neurallabs.net/es/soluciones/vpar-server.

[15] COCO-Text: Dataset for Text Detection and Recognition | SE(3) Computer Vision Group at Cornell Tech. (n.d.). Retrieved October 1, 2021, from https://vision.cornell.edu/se3/coco-text-2/.

[16] The Street View Text Dataset (SVT) - TC-11. (n.d.). Retrieved October 1, 2021, from http://tc11.cvc.uab.es/datasets/SVT_1.

[17] GitHub - ankush-me/SynthText: Code for generating synthetic text images as described in "Synthetic Data for Text Localisation in Natural Images", Ankush Gupta, Andrea Vedaldi, Andrew Zisserman, CVPR 2016. (n.d.). Retrieved October 1, 2021, from https://github.com/ankush-me/SynthText.

[18] Reconocimiento de Dígitos (Números) Manuscritos. Cursos de Programación de 0 a Experto © Garantizados. (n.d.). Retrieved October 1, 2021, from https://unipython.com/reconocimiento-de-digitos-numeros-manuscritos/.[19] tesseract-ocr · GitHub. (n.d.). Retrieved October 1, 2021, from https://github.com/tesseract-ocr/.

[20] Poppler. (n.d.). Retrieved October 4, 2021, from https://poppler.freedesktop.org/

[21] XpdfReader. (n.d.). Retrieved October 4, 2021, from http://www.xpdfreader.com/

[22] pdf2image · PyPI. (n.d.). Retrieved October 26, 2021, from https://pypi.org/project/pdf2image/.

[23] ImageMagick – Convert, Edit, or Compose Digital Images. (n.d.). Retrieved October 4, 2021, from https://imagemagick.org/index.php.

[24] Recursos – ImageMagick. Articulo de Revista. Retrieved October 4, 2021, from https://www.uv.es/scubero/recursos/imagemagik-cli.pdf.

[25] Wand — Wand 0.5.8. (n.d.). Retrieved October 26, 2021, from https://docs.wand-py.org/en/0.5.8/index.html.

[26] Wand·PyPI. (n.d.). Retrieved December 12, 2021, from https://pypi.org/project/Wand/.

[27] Math — Mathematical functions — Python 3.10.1 documentation. (n.d.). Retrieved December 12, 2021, from https://docs.python.org/3/library/math.html.

[28] GIMP - GNU Image Manipulation Program. (n.d.). Retrieved October 4, 2021, from https://www.gimp.org/.

[29] OCR Application For Enterprise | Premium OCR |Kofax OmniPage. (n.d.). Retrieved October 6, 2021, from https://www.kofax.com/products/omnipage?source=nuance.

[30] Readiris 17, la solución PDF y OCR para Windows. (n.d.). Retrieved October 6, 2021, from https://www.irislink.com/ES/c1729/Readiris-17--la-solucion-PDF-y-OCR-para-Windows-.aspx.

[31] PDF Software: Open, Read & Edit PDF | FineReader PDF. (n.d.). Retrieved October 6, 2021, from https://pdf.abbyy.com/es/.

[32] gImageReader download | SourceForge.net. (n.d.). Retrieved October 6, 2021, from https://sourceforge.net/projects/gimagereader/.

[33] Best OCR to Word Software to Extract Text from Image to Save as Word. (n.d.). Retrieved October 6, 2021, from https://www.ocrtoword.com/.

[34] GitHub - ocropus/ocropy: Python-based tools for document analysis and OCR. (n.d.). Retrieved October 6, 2021, from https://github.com/ocropus/ocropy.

[35] Cuneiform for Linux in Launchpad. (n.d.). Retrieved October 6, 2021, from https://launchpad.net/cuneiform-linux.

[36] Ocrad - GNU Project - Free Software Foundation (FSF). (n.d.). Retrieved October 6, 2021, from https://www.gnu.org/software/ocrad/.

[37] GOCR. (n.d.). Retrieved October 6, 2021, from http://jocr.sourceforge.net/.

[38] GitHub - tesseract-ocr/tesseract: Tesseract Open Source OCR Engine (main repository). (n.d.). Retrieved October 6, 2021, from https://github.com/tesseract-ocr/tesseract.

[39] Dhiman, S., & Singh, A. J. (2013). Tesseract Vs Gocr A Comparative Study. International Journal of Recent Technology and Engineering (IJRTE), 2, 2277–3878. http://code.google.com/p/tesseract-ocr.

[40] Wordle. (n.d.). Retrieved October 7, 2021, from http://www.wordle.net/.

[41] Wordaizer | word cloud with a twist. (n.d.). Retrieved October 7, 2021, from https://www.apphelmond.com/Wordaizer/.

[42] Tagxedo - Word Cloud with Styles. (n.d.). Retrieved October 7, 2021, from http://www.tagxedo.com/.

[43] pytagcloud·PyPI. (n.d.). Retrieved October 7, 2021, from https://pypi.org/project/pytagcloud/.

[44] Wordle. (n.d.). Retrieved October 27, 2021, from http://www.wordle.net/

[45] python - How do I install pip on macOS or OS X? - Stack Overflow. (n.d.). Retrieved October 7, 2021, from https://stackoverflow.com/questions/17271319/how-do-i-install-pip-on-macos-or-os-x.

[46] NLTK :: Natural Language Toolkit. (n.d.). Retrieved November 23, 2021, from https://www.nltk.org/.

[47] NLTK Book. (n.d.). Retrieved November 23, 2021, from https://www.nltk.org/book/.

[48] Gensim: Topic modelling for humans. (n.d.). Retrieved October 27, 2021, from https://radimrehurek.com/gensim/index.html#install.

[49] Azure Machine Learning - ML as a Service | Microsoft Azure. (n.d.). Retrieved October 7, 2021, from https://azure.microsoft.com/en-us/services/machine-learning/.

[50] Pricing – Machine Learning Studio (Classic) | Microsoft Azure. (n.d.). Retrieved October 7, 2021, from https://azure.microsoft.com/en-us/pricing/details/machine-learning-studio/.

[51] Amazon SageMaker – Aprendizaje automático – Amazon Web Services. (n.d.). Retrieved October 7, 2021, from https://aws.amazon.com/es/sagemaker/.

[52] Amazon SageMaker – Aprendizaje automático – Amazon Web Services. (n.d.). Retrieved October 7, 2021, from https://aws.amazon.com/es/sagemaker/prices/.

[53] Machine Learning as a Service – MLaaS - Data Science Central. (n.d.). Retrieved October 7, 2021, from https://www.datasciencecentral.com/profiles/blogs/machine-learning-as-a-service-mlaas.

[54] Watson Studio | IBM. (n.d.). Retrieved October 7, 2021, from https://www.ibm.com/cloud/watson-studio.

[55] Watson Studio - Pricing | IBM. (n.d.). Retrieved October 7, 2021, from https://www.ibm.com/cloud/watson-studio/pricing.

[56] Vertex AI | Vertex AI | Google Cloud. (n.d.). Retrieved October 7, 2021, from https://cloud.google.com/vertex-ai.

[57] Resumen de precios | Google Cloud. (n.d.). Retrieved October 7, 2021, from https://cloud.google.com/pricing/.

[58] BigML.com. (n.d.). Retrieved October 7, 2021, from https://bigml.com/.

[59] About BigML.com. (n.d.). Retrieved October 7, 2021, from https://bigml.com/about.

[60] Pricing | BigML.com. (n.d.). Retrieved October 7, 2021, from https://bigml.com/pricing#subscriptions.

[61] Dataiku | Everyday AI, Extraordinary People. (n.d.). Retrieved October 7, 2021, from https://www.dataiku.com/.

[62] PyCharm: el IDE de Python para desarrolladores profesionales, por JetBrains. (n.d.). Retrieved October 27, 2021, from https://www.jetbrains.com/es-es/pycharm/.

[63] ImageMagick – Download. (n.d.). Retrieved October 30, 2021, from https://imagemagick.org/script/download.php.

[64] Home · UB-Mannheim/tesseract Wiki. (n.d.). Retrieved October 30, 2021, from https://github.com/UB-Mannheim/tesseract/wiki.

[65] Downloading and installing Weka - Weka Wiki. (n.d.). Retrieved October 30, 2021, from https://waikato.github.io/weka-wiki/downloading_weka/.

Presupuesto

Este capítulo contiene el presupuesto que recoge los gastos generados en la realización del presente TFG.

P.1 Componentes del presupuesto

El presupuesto calculado se divide en las siguientes partes:

- Recursos materiales.
- Trabajo tarifado por tiempo empleado.
- Material Fungible.
- Redacción de la documentación.
- Derechos de visado del *Colegio Oficial de Ingenieros Técnicos de Telecomunicación* (*COITT*).
- Gastos de tramitación y envío.
- Aplicación de impuestos y coste total.

P.2 Recursos Materiales

Para la correcta ejecución y desarrollo del TFG han sido necesarios numerosos recursos hardware, autorizaciones de autoridades para la recepción y trabajo con sus documentos y distintas herramientas software, las cuales pueden llevar asociadas un coste en sus licencias. Entre estos recursos, se podrían destacar el paquete de Microsoft Office para la redacción de esta memoria, los distintos ordenadores con diferentes Sistemas Operativos (Linux, MacOS y Windows).

Así cabe destacar que la mayor parte del software utilizado no conlleva un pago de licencia, ya que es una de las condiciones que hemos llevado a cabo, a excepción del Microsoft Office.

Con el fin de establecer un cálculo de la amortización, se presupone el sistema de amortización como lineal, de tal forma que se asume que el inmovilizado material se desprecia de forma constante a lo largo de su vida útil.

Así, para llevar a cabo el cálculo de la cuota de amortización anual, se calcula usando la siguiente ecuación.

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil}$$

Ecuación P1.

Recurso	Unidades	Valor de adquisición (€)	Valor residual (*) (€)	Vida útil (años)	Cuota anual (€)	Uso (meses)	Cuota aplicable (€)
HP Pavilion Laptop 15-cs3xxx Procesador Intel(R) Core (TM) i7-1065G7 16,0 GB RAM Sistema operativo de 64 bits.	1	800	240	5	112	5	46,67
Mac mini Apple (2020) MGNT3Y/A, Apple Silicon Chip M1, 8 GB, 512 GB SSD	1	900	270	5	126	5	52,5
Monitor - Lenovo Q24I-1L, 23.8 FHD, 4 ms, 75 Hz, NTSC 72 %, 1000:1, AMD FreeSync™, 2 altavoces	1	179	0	5	35.8	5	60
Licencia Paquete Office Windows (365)	1	99	0	1	99	5	41.25
MATERIALES AMORTIZABLES	**TOTAL**						200.42

Tabla P1. Amortización Total

En cuanto a la amortización total aparece reflejada en la Tabla P.1, así como los diferentes valores y cuotas de los diferentes recursos empleados en este TFG.

El valor residual puede ser definido como el valor teórico supuesto que tendría el elemento después de su vida útil. En el caso de los portátiles un terminal móvil y portátil se deprecian en una tasa del 30 % con respecto a su valor. Según Artículo 34, LISR.

El coste de los materiales amortizables es de un total de: doscientos euros con 42 céntimos (200.42 €).

P.3 Trabajo tarificado por tiempo empleado

Este concepto contabiliza los gastos que corresponden a la mano de obra, según el salario correspondiente a la hora de trabajo de un Ingeniero Técnico de Telecomunicación.

Según la tabla retributiva de personal contratado en proyectos de investigación elaborada por la ULPGC en el año 2016, este salario, con una dedicación de 20 horas semanales, asciende a 896,31 € mensuales. Lo que se aproxima a una retribución de 11,20 €/h.

Este TFG tal como comprende el Proyecto Docente ha conllevado 300 horas, por lo que se calcula el coste total por tiempo empleado en:

"11,20 · 300 = 3360,00 €"

Por lo tanto, el trabajo tarifado por tiempo empleado asciende a la cantidad de tres mil trescientos sesenta euros (3.360,00 €).

P.4 Redacción del trabajo

Se ha utilizado la Ecuación P.2 para determinar el coste asociado a la redacción de la memoria del presente TFG.

$$R = 0{,}07 \times P \times Cn$$

Ecuación P.1

Donde:

- R son los honorarios por la redacción del trabajo.
- P es el presupuesto.
- Cn es el coeficiente de ponderación en función del presupuesto.

El valor del presupuesto P se calcula sumando los costes del trabajo tarifado por tiempo empleado y de la amortización del inmovilizado material, tanto hardware como software. El resultado de los costes se muestra en la Tabla P.2.

Como el coeficiente de ponderación Cn para presupuestos menores de 30.050,00 € viene definido por el COITT con un valor de 1.00, el coste derivado de la redacción del TFG es de:

$$R = 0{,}07 \times 3560.42\,€ \times 1 = 249{,}23\,€$$

Ascendiendo de esta forma el coste de la redacción del trabajo a ***doscientos cuarenta y nueve euros con veinte y tres céntimos*** (249,23 €).

Concepto	Coste (€)
Trabajo tarifado por tiempo empleado	3360,00
Amortización del material	200,42
Total	3560,42

Tabla P.2. Valor del presupuesto

P.5 Material Fungible

No se contempla ningún gasto por edición de documentos ni por gastos de material de oficina, aunque en este proyecto se editan los documentos recibidos, como dichos documentos son ofrecidos por el interesado no presentamos ningún tipo de coste al respecto, por lo que el coste asociado al material fungible es de cero euros (0 €).

P.6 Derechos de visado del COITT

El COITT establece que, para proyectos técnicos de carácter general, los derechos de visado para 2016 se calculan en base a la Ecuación P.3:

$$V = 0.006 * P_1 * C_1 + 0.003 * P_2 * C_2$$

Ecuacion P.3

Donde:

- V es el coste de visado del trabajo.
- P1 es el presupuesto del proyecto.
- C1 es el coeficiente reductor en función del presupuesto.
- P2 es el presupuesto de ejecución material correspondiente a la obra civil.
- C2 es el coeficiente reductor en función a P2.

El valor del presupuesto P1 se halla sumando los costes de las secciones correspondientes al trabajo tarifado por tiempo empleado, a la amortización del inmovilizado material y a la redacción del documento. Esta suma se muestra en la Tabla P.3. Al igual que en el caso anterior, el coeficiente C1 para proyectos de presupuesto inferior a 30.050,00 € es de 1,00 €, asimismo el valor de P2 es de 0,00 € ya que no se realiza ninguna obra.

Concepto	Coste (€)
Trabajo tarifado por tiempo empleado	3360.00
Amortización del material	200.42
Redacción del trabajo	249.23
Total	3809.65

Tabla P3. Presupuesto P_1

De esta forma, aplicando a la Ecuación P.3 los datos de la Tabla P.4 y el coeficiente especificado se obtiene:

$$V=0{,}006\cdot 3809{,}65\cdot 1=22{,}86\,€$$

Los costes por derechos de visado del presupuesto ascienden a veintidós euros con ochenta y seis céntimos (22.13 €).

P.7 Gastos de tramitación y envío

Los gastos de tramitación y envió están estipulados en seis euros (6,00 €) por cada documento visado de forma telemática.

P.8 Aplicación de impuestos y coste total

El presupuesto total del presente TFG está gravado por el Impuesto General Indirecto Canario (*IGIC*), que está establecido en la actualidad en un siete por ciento (7 %). El coste total de este TFG se encuentra desglosado en la Tabla P.5.

Descripción	Subtotal (€)
Amortización de materiales	200,42
Trabajo tarifado por tiempo empleado	3360,00
Costes de material fungible	0
Redacción del trabajo	249,23
Derechos de visado del COIT	22.13
Gastos de tramitación y envío	6,00
Suma (€)	3837,78
IGIC (7%)	268,64
TOTAL	4106,42

Tabla P.5. Coste total

El presupuesto total del proyecto *Aplicación de algoritmos de Machine Learning para la clasificación documental* asciende a: **cuatro mil ciento seis euros con cuarenta y dos céntimos** (4106.42 €).

Fdo.: Erik Martrus Guillén

En Las Palmas de Gran Canaria a …… de …….. de ……

Pliego de condiciones

En este episodio se detallan los aspectos relacionados con la información de licencias, los derechos de autoría y las responsabilidades. Así como las condiciones bajo las que se ha desarrollado este TFG y las que se establecen al usuario para poder usar la plataforma que se ofrece.

PL1. Condiciones Hardware

Durante el desarrollo de este TFG se han usado los dispositivos hardware recogidos en la Tabla PL.1.

PL2. Condiciones Software

En la Tabla PL.2 se exponen las herramientas software utilizadas, especificando su versión.

Equipo	Modelo	Fabricante
Ordenador portátil	Pavilion 15	HP
Mini Mac	Mac Mini	Apple
Pantalla Monitor	Lenovo Q24I	Lenovo

Tabla PL.1. Condiciones del Hardware

Aplicación	Versión
biblioteca OS	3.9.5
Weka	3.9.5
NLTK	3.6.5
Wordcloud	1.8.1
Pytesseract	0.3.8
Typora	0.11.13
Microsoft Office	365
GIMP	2.10.28
PyCharm Community Edition	2019.3.3
ImageMagick	Wand 0.6.7
Python	3.6
Poppler	python-poppler 0.2.2
MacOs	macOS 10.15 Catalina
Windows	10

Tabla PL.2. Condiciones del Software

PL3. Condiciones de licencia

El código desarrollado en este trabajo es propiedad de la Universidad de Las Palmas de Gran Canaria y todos los que pretendan hacer uso de él deben aceptar todas las cláusulas establecidas en esta licencia. El uso de las plataformas se podría hacer bajo autorización del autor, tutores del TFG, y la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

PL4. Derechos de autor

Tanto el código como la plataforma e información que se adjunta están protegidos por las leyes de propiedad intelectual que les sean aplicables, así como las disposiciones de los tratados internacionales. Por tanto, el código se considera un producto protegido por derechos de autor. Esto no es óbice para que una persona pueda copiar o utilizar el código bajo la autorización del autor, tutores del TFG, y la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

PL5. Restricciones

No se permite el uso de ingeniería inversa. Sí se permite la transferencia del código a un tercero siempre que no se conserve ninguna copia, incluyendo actualizaciones o material escrito adicional.

PL6. Garantía

El autor del TFG lo presenta AS IS (tal cual), sin garantía implícita de ningún tipo. No se responsabiliza de los daños que pudieran causar a equipos o personas por el uso del código o la documentación. El autor no asegura, garantiza, o realiza ninguna declaración sobre el uso y resultados derivados de la utilización del código y/o del resto de la información proporcionada.

El código y la plataforma elaborada no están exentos de errores y no está diseñado para entornos de riesgo que requieran de un funcionamiento a prueba de

fallos. El autor rechaza expresamente cualquier garantía explícita e implícita de adecuación del código para actividades de riesgo.

PL7. Limitación de responsabilidad

En ningún caso el autor ni los tutores, ni la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de las Palmas de Gran Canaria serían responsables de los perjuicios directos, indirectas incidentales o consiguientes, gastos, lucro cesante, pérdida de ahorros, interrupción de negocios, pérdida de información comercial o de negocio, o cualquier otra pérdida que resulte del uso o de la incapacidad de usar el código o la documentación. El usuario conoce y acepta este riesgo, así como el resto de las cláusulas y restricciones. El autor no reconoce otra garantía que no haya sido indicada anteriormente.

PL8. Otras consideraciones

En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente inválida, las cláusulas afectadas serían modificadas convenientemente de manera que sean ejecutables una vez modificadas, permaneciendo el resto de este contrato en vigencia. Este contrato se rige por las leyes de España. El usuario acepta la jurisdicción exclusiva de los tribunales españoles con relación a las disputas derivadas de la presente licencia.

ANEXOS

A. Descargo de responsabilidad

Descargo de responsabilidad: como bien se ha dicho a lo largo del documento, los documentos que recibimos como entrada y que serán los que conformen nuestro banco de datos provienen de una entidad oficial como es Autoridad Portuaria de Santa Cruz de Tenerife, estos documentos fueron facilitados por Avantic, empresa donde realice la practicas de empresa.

Este descargo de responsabilidades permite garantizar la seguridad y privacidad del contenido de los documentos así de cómo Nombres, Apellidos, o material sensible de cualquier otra índole.

B. Herramientas de OCR

Aunque usamos Tesseract resumimos (Tabla B.1) las distintas herramientas que probamos.

Como ya hemos mencionado anteriormente se descartarán en primer lugar los sistemas que trabajan con un único Sistema Operativo, como pueden ser Cuneiform OCR y OCRopus y además también aquellos sistemas que sean de pago que a menudo implica que los documentos puedan ser tratados con OCR en la nube, vulnerando así la política de privacidad para dichos documentos.

Nombre	SO	CMD/Terminal	En la nube/Local	De Pago/Gratis	Resultado
Nuance OmniPage	Todos	No	Nube	Pago	-
ABBYY Fine Reader OCR	Todos	Si	Nube	Pago	-
GOCR	Todos	Si	Local	Gratis	Bueno
GNU Ocrad	Todos	Si	Local	Gratis	Bueno
Tesseract	Todos	Si	Local	Gratis	Bastante Bueno
Cuneiform OCR	Linux	Si	Local	Gratis	-
OCRopus	Linux	Si	Local	Gratis	-

Tabla B.1. Herramientas OCR

C. Herramientas de Machine Learning

En este anexo se muestra las elecciones de las distintas tecnologías de ML que se implementaron durante este proyecto y se explica el porqué de estas elecciones.

En primer lugar, se disponían de dos herramientas para la implementación de la IA en nuestro proyecto, Google Cloud e IBM Watson Machine Learning. En la elección de este me he decantado por la opción de utilizar Weka finalmente ya que se realizaron diversas pruebas utilizando el servicio de IBM, pero ninguna dio resultados satisfactorios.

Por otro lado, si bien con la implementación de Google Cloud pudimos obtener resultados de clasificación como tal, pero las salidas que obteníamos a dicha clasificación y la implementación de la API que implica Tensorflow no aportan resultados válidos y fiables en lo que respecta al objetivo buscado.

Hay que destacar, que como ya hemos mencionado, la facilidad y sencillez de Weka además de su eficacia y un nivel de gran detalle en cuanto a los resultados nos han llevado a cogerla como herramienta para esta parte del proyecto, facilitando así todas estas tareas durante el momento del confinamiento ocurrido en España debido al COVID-19 de una manera muy satisfactoria.

D. Ejemplos de OCR

Algunos ejemplos de OCR y del texto plano conseguido como salida.

Cada vez más cerca de las personas

Cruz Roja Española

Comité provincial S/C de Tenerife

Cruz Roja Española
Asamblea Provincial de S/C. Tenerife
26 DIC 2019
S A L I D A
Núm. 2039

Sr. D. Aitor Acha Bedialauneta
Director de la Autoridad Portuaria de Santa Cruz de Tenerife
Avda. Fco. La Roche, 49 Edificio Junta del Puerto
38001- Santa Cruz de Tenerife

Universalidad Unidad Voluntariado Independencia Neutralidad Imparcialidad Humanidad

Santa Cruz de Tenerife, a lunes 23 de diciembre de 2019.

Estimado Sr.

En nombre de Cruz Roja y en el mío propio, queremos hacerle llegar nuestro más sincero agradecimiento a Ud., y a todo su equipo por concedernos un año más la autorización para la utilización de la nave ubicada en el muelle Ribera del Puerto de Santa Cruz de Tenerife (antiguo Consorcio), cedida por ustedes hasta el próximo día 31 de diciembre de 2020, omitiendo el abono de las tasas de ocupación, y de actividad.

La autorización de este espacio nos permitiría seguir facilitando la distribución de material y mercancía para fines sanitarios y humanitarios de la Institución, así como el almacenaje de material de salvamento marítimo.

Reciba un cordial y afectuoso saludo.

OFICINA PROVINCIAL DE CRUZ ROJA ESPAÑOLA de Tenerife

María Teresa Pociello Marty
Presidenta Provincial de Cruz Roja en S/C de Tenerife

C/ San Lucas, 60. 38002 Santa Cruz de Tenerife
Tfno.: 922.282.924 Fax 922.246.744
CIF Q2866001G

Este documento no dispone de firma electrónica.
Registro: REG-E-GEN-2020/000093-09/01/2020 08:40:00

Página

Figura D.1. Ejemplo de imagen obtenida del PDF

3
0 Santa Cruz de Tenerife, a lunes 23 de diciembre de 2019.
B
Cc
> .

Estimado Sr.
°o
Cv
©
2 En nombre de Cruz Roja y en el mio propio, queremos hacerle llegar nuestro mas
2 sincero agradecimiento a Ud., y a todo su equipo por concedernos un afio mas la
> autorizacion para la utilizacion de la nave ubicada en el muelle Ribera del Puerto de

Santa Cruz de Tenerife (antiguo Consorcio), cedida por ustedes hasta el proximo dia
© 31 de diciembre de 2020, omitiendo el abono de las tasas de ocupacion, y de
c actividad.
2
g La autorizacién de este espacio nos permitiria seguir facilitando la distribucion de
3 material y mercancia para fines sanitarios y humanitarios de la Institucién, asi como el
= almacenaje de material de salvamento maritimo.
8
z Reciba un cordial y afectuoso saludo.
o
f

Figura D.2. Salida del OCR obtenida en texto plano