UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Neural network classification of handwritten document images based on probabilistic indexing

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Juan José Flores Arellano
Tutor: Enrique Vidal Ruiz
Cotutor: Jose Ramón Prieto Fontcuberta

Curso 2020-2021

Agradecimientos

A Enrique y Jose Ramón por la paciencia e implicación en el proyecto.

A mi familia por el apoyo recibido durante los cuatro años de grado.

Resum

La classificació de documents manuscrits basada en el contingut és una important tasca que generalment es realitza en arxius i biblioteques per experts amb un gran coneixement sobre el contingut dels documents. Però, desafortunadament, moltes col·leccions de manuscrits són tan vastes que no és factible dependre únicament d'experts per a fer aquesta tasca. Els enfocaments actuals per a la classificació de manuscrits basada en el contingut textual generalment requereix que les imatges de text manuscrit siguen transcrites i convertides en text electrònic. Però per a grans col·leccions de manuscrits històrics la transcripció manual és generalment inviable. I a causa de les imprecisions inherents al text, incerteses degudes a lèxic arcaic i estat de conservació dels documents, la transcripció automàtica no aconsegueix obtindre resultats prou precisos.

En aquest projecte es proposa un nou enfocament per a fer aquesta tasca de classificació que no requereix de transcripcions explícites de les imatges. Es basa en 'indexació probabilística', una tecnologia relativament nova que permet representar eficaçment les incerteses intrínseques que estan generalment presents en els textos manuscrits històrics. Es proposa treballar sobre lligalls del segle XVII de l'Arxiu Històric Provincial de Cadis. Cada lligall conté centenars d'expedients notarials manuscrits de diverses tipologies (Venda, Arrendament, Poder, Testament, etc.). L'objecte és classificar cada expedient en la seua corresponent tipologia. Un sistema que resolga satisfactòriament aquesta tasca té una enorme aplicabilitat en centenars, o milers, d'arxius i biblioteques que custodien milions de documents que no han pogut ser catalogats adequadament a causa de l'enorme envergadura de la tasca per a un processament purament manual.

D'altra banda, la metodologia a desenvolupar en aquest projecte pot obrir portes per a abordar moltes altres tasques d'analítica de text sobre grans volums d'imatges de text manuscrit sense transcriure. Per tant, aquests desenvolupaments tenen també un gran interés cientificotècnic i poden donar lloc a publicacions acadèmiques rellevants.

Com a conclusió, assenyalar que tot el codi desenvolupat durant el projecte serà dipositat en un repositori públic, amb l'objectiu que futurs treballs puguin continuar des del fet en aquest.

Paraules clau: Documents manuscrits, Reconeixement d'images, Classificació de documents, Indexació probabilísitica.

Resumen

La clasificación de documentos manuscritos basada en el contenido es una importante tarea que generalmente se realiza en archivos y bibliotecas por expertos con un gran conocimiento sobre el contenido de los documentos. Pero, desafortunadamente, muchas colecciones de manuscritos son tan vastas que no es factible depender únicamente de expertos para realizar esta tarea. Los enfoques actuales para la clasificación de manuscritos basada en el contenido textual generalmente requiere que las imágenes de texto manuscrito sean transcritas y convertidas en texto electrónico. Pero para grandes colecciones de manuscritos históricos la transcripción manual es generalmente inviable. Y debido a las imprecisiones inherentes al texto, incertidumbres debidas a léxico arcaico y estado de conservación de los documentos, la transcripción automática no consigue obtener resultados suficientemente precisos.

En este proyecto se propone un nuevo enfoque para realizar esta tarea de clasificación que no requiere de transcripciones explícitas de las imágenes. Se basa en 'indexación probabilística', una tecnología relativamente novedosa que permite representar eficazmente las incertidumbres intrínsecas que están generalmente presentes en los textos manuscritos históricos. Se propone trabajar sobre legajos del siglo XVII del Archivo Histórico Provincial de Cádiz. Cada legajo contiene centenares de expedientes notariales manuscritos de diversas tipologías (Venta, Arrendamiento, Poder, Testamento, etc.). El objeto es clasificar cada expediente en su correspondiente tipología. Un sistema que resuelva satisfactoriamente esta tarea tiene una enorme aplicabilidad en cientos, o miles, de archivos y bibliotecas que custodian millones de documentos que no han podido ser catalogados adecuadamente a causa de la enorme envergadura de la tarea para un procesado puramente manual.

Por otra parte, la metodología a desarrollar en este proyecto puede abrir puertas para abordar muchas otras tareas de analítica de texto sobre grandes volúmenes de imágenes de texto manuscrito sin transcribir. Por tanto, estos desarrollos tienen también un gran interés científico-técnico y pueden dar lugar a publicaciones académicas relevantes.

Como conclusión, señalar que todo el código desarrollado durante el proyecto será depositado en un repositorio público, con el objetivo de que futuros trabajos puedan continuar desde lo hecho en este.

Palabras clave: Documentos manuscritos, Reconocimiento de imágenes, Clasificación de documentos, Indexación probabilística.

Abstract

The classification of manuscript documents based on content is an important task that is usually performed in archives and libraries by experts with a great deal of knowledge about the content of the documents. usually performed in archives and libraries by experts with a great deal of knowledge about the content of the documents. But, unfortunately, many manuscript collections are so vast that it is not feasible to rely solely on experts to perform this task. Current approaches to manuscript classification based on textual content generally require manuscript text images to be transcribed and converted into electronic text. But for large collections of historical manuscripts manual transcription is generally infeasible. And due to inherent textual inaccuracies, uncertainties due to archaic lexicon and state of preservation of the documents, automatic transcription fails to obtain sufficiently accurate results.

This project proposes a new approach to this classification task that does not require explicit transcriptions of the images. requires explicit transcriptions of the images. It is based on 'probabilistic indexing', a relatively novel technology that allows to efficiently represent the intrinsic uncertainties that are generally present in historical manuscript texts. that are generally present in historical manuscript texts. It is proposed to work on 17th century files from the Provincial Historical Archive of Cadiz. Each file contains hundreds of handwritten notarial records of various types (Sale, Lease, Power of Attorney, Will, etc.). The objective is to classify each file in its corresponding typology. A system that satisfactorily solves this task has an enormous applicability in hundreds, or thousands, of archives and hundreds, or thousands, of archives and libraries that hold millions of documents that have not been properly catalogued because of the enormous size of the task for a purely manual processing. purely manual processing

On the other hand, the methodology to be developed in this project may open doors to address many other text analytical tasks on large volumes of untranscribed many other text analytics tasks on large volumes of untranscribed manuscript text images. Therefore, these developments are also of great scientific-technical interest and may lead to relevant academic publications.

In conclusion, all the code developed during the project will be deposited in a public repository, so that future work can continue from what has been done in this project.

Key words: Manuscript documents, Image recognition, Documents classification, Probabilistic indexing.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El documento manuscrito es aquel documento que contienen información escrita a mano en un soporte flexible y manejable (ej. pergamino, papiro, papel), con materias como la tinta de pluma, de un bolígrafo o de un lápiz de grafito.

Desde la antigüedad, este medio se utiliza para almacenar y transmitir conocimientos, relatos o creencias a sus coetáneos, a las siguientes generaciones o a otras culturas [1].

Hoy en día contamos con unas cantidades ingentes de documentos manuscritos, con información muy valiosa sobre la historia del ser humano y sus avances en diferentes campos como la política y la cultura.

Pero por desgracia, la información o contenido textual de estos documentos permanece inaccesible para la extracción de la información que estos poseen. Esto es debido a la difícil y costosa tarea de la transcripción de los mismos, que hasta ahora, se realizaban manualmente por expertos altamente cualificados. Es por esto que se acababa convirtiendo en una tarea algo lento y cara, y que solo se realizaba a conjuntos pequeños de documentos manuscritos, ya que para grandes colecciones, era algo inasequible para archivos y bibliotecas.

Actualmente existen tecnologías como OCR (del inglés, Optical Character Recognition) e ICR (del inglés, Intelligent Character Recognition), que permiten el reconocimiento de caracteres aislados, impresos y manuscritos, a partir de la digitalización del documento. Estas tecnologías, requieren un proceso previo de segmentación y extracción para la transcripción de palabras o lineas.

Esto último limita la eficiencia del sistema, especialmente para la tarea que estamos tratando (reconocimiento de texto manuscrito basado en el contenido), ya que en algunos casos es imposible el aislar correctamente los caracteres.

Lo que pretendemos hacer en este proyecto, es algo similar a la conocida tarea de 'clasificación de documentos basados en su contenido manuscrito' que parte de la base de que los datos son documentos de texto plano en lugar de imágenes de documentos de textos manuscritos.

La clasificación de documentos basados en su contenido, parte de la digitalización como un primer paso y continua con el uso de 'índices probabilísticos', una tecnología novedosa que nos permitirá hacer frente a la incertidumbre intrínseca que generalmente presentan las palabras escritas a mano en los documentos.

1.1 Motivación

La utilidad de la clasificación de textos manuscritos en base a su contenido, radica principalmente en encontrar la información que buscamos sobre algún hecho histórico de una forma mucho más rápida y eficiente de lo que se hace hasta ahora, además de conocer el contenido o tema del documento previamente a su lectura.

En este trabajo se propone resolver la clasificación de los textos manuscritos en base a su contenido, los documentos en cuestión son legajos del Archivo Provincial de Cádiz del siglo XVII, en los que cada legajo posee actas notariales las cuales pertenecen a una de las nueve clases diferentes.

El desarrollo del proyecto se puede dividir en tres fases: selección de características mediante *Information Gain* (IG a partir de ahora), cálculo de los valores de las características seleccionadas mediante Tf·Idf (del inglés Term Frequency - Inverse Document Frequency) y la clasificación de los propios documentos empleando Redes Neuronales Artificiales. Más exactamente, emplearemos diferentes variantes del Perceptrón Multicapa (MLP de sus siglas en inglés Multi Layer Perceptron).

La estimación del *Information Gain* nos será útil para realizar un ranking de las palabras que nos aportan más información acerca de la clase del documento en el que se encuentra la misma, y el cálculo del Tf·Idf nos será de gran ayuda para saber como de relevante es una palabra para un documento en una colección.

1.2 Objetivos

La principal finalidad de este proyecto es la de la exploración de la reciente tecnología de los índices probabilísticos así como de afianzarla en el panorama del Machine Learning y de la clasificación de documentos con texto manuscrito. Para esto, nos apoyaremos en el uso de Redes Neuronales Artificiales para la clasificación de documentos manuscritos en base a su contenido.

Otro de los objetivos, es el de la exploración del legajo JMDB_4949 de la colección del Archivo Provincial de Cádiz, a la par de la clasificación de los expedientes que se encuentran en este legajo.

Para ello, se pretende crear un sistema capaz de tomar los índices probabilísticos de cada documento o expediente perteneciente a la colección y realizar una ordenación de pseudo-palabras del corpus en base a su *Information Gain*. Este será, un sistema eficiente, utilizando las estructuras de datos adecuadas en cada momento, siendo capaz de procesar cientos de ficheros en tiempos razonables.

Después, se calculará el Tf·Idf de cada palabra de cada documento de la colección, para posteriormente entrenar un clasificador. El objetivo es el de clasificar los documentos manuscritos en una de las 9 clases presentes en la colección.

1.3 Trabajos Relacionados

En cuanto a trabajos relacionados, cabe decir que la tarea a realizar en este trabajo es muy diferente a las tareas de visión por computador y análisis de imágenes, comúnmente llamada 'clasificación de imágenes' [2],[3],[4]. En dichas tareas las imágenes son clasificadas según sus características globales relacionadas con los colores, texturas, formas, etc.

Por otro lado, destacar que es muy diferente a la tarea de 'clasificación de imágenes basada en su contenido' [5], [6], ya que en una tarea convencional de clasificación de imágenes basada en su contenido, las imágenes suelen contener pocos objetos y relativamente grandes como pueden ser montañas, vehículos, personas...

En cambio, en este trabajo trataremos con imágenes de texto las cuales contienen varios cientos de objetos o palabras y además, algunos de estos objetos están poco detallados debido a que son documentos manuscritos antiguos y, por tanto, existen imperfecciones en los mismos. Es por esto que trabajos como los citados anteriormente no son comparables con el trabajo que se presenta a continuación.

Podemos encontrar un trabajo anterior que está fuertemente ligado y relacionado con lo hecho en este proyecto, [7]. En este, al igual que aquí, se utilizan índices probabilísticos para la clasificación de textos manuscritos en base a su contenido en tres clases (Alto, Medio y Bajo), dependiendo del riesgo de que el documento caiga en manos de caza tesoros.

Es por esto que este proyecto se basa en el anterior trabajo con la idea de afianzar la reciente tecnología de los índices probabilísticos. Además, en el presente trabajo exploraremos un corpus diferente.

1.4 Estructura de la Memoria

En cuanto a la estructura de memoria del trabajo, continuaremos la propia memoria con el Capítulo 2 - Fundamentos, donde explicaremos el tipo de redes neuronales artificiales utilizadas en el proyecto, al igual que el origen y uso de los índices probabilísticos. En este capítulo también expondremos tanto el IG como el Tf·Idf, además de analizar como la indexación probabilística nos ayuda a estimar las distintas características que un documento manuscrito posee.

A continuación, en el Capítulo 3 - Trabajo realizado, explicaremos como hemos resuelto las ecuaciones para la estimación de *Information Gain* y Tf·Idf además de introducir la arquitectura implementada para predecir la clase a la que pertenece cada documento de la colección.

Posteriormente, en el Capítulo 4 - Corpus, explicaremos el origen de dicho corpus con el cual hemos trabajado durante el proyecto, al igual que su procedencia y utilidad.

Continuaremos la memoria del trabajo con el Capítulo 5 - Experimentos y resultados, donde como el propio nombre del capítulo enuncia, exploraremos los detalles de los distintos experimentos realizados a lo largo del proyecto así como los resultados obtenidos de los mismos.

Tras los capítulos anteriormente descritos, llegaremos al Capítulo 6 - Conclusiones, donde discutiremos los resultados obtenidos, además de proponer posibles trabajos relacionados que puedan solventar o paliar las carencias y errores que hemos podido realizar a lo largo de la resolución del problema.

CAPÍTULO 2
Fundamentos

A lo largo de este capítulo exploraremos las distintas tecnologías y métricas utilizadas, además de justificar su uso y origen.

2.1 Redes Neuronales Artificiales (ANN)

Las redes neuronales artificiales o ANN (del inglés Artificial Neural Network) son un modelo computacional basado en las redes neuronales biológicas. Consisten en un conjunto de unidades llamadas *neuronas*, conectadas entre sí para transmitir información. Dicha información atraviesa la red sometiéndose a diversas operaciones produciendo así un *output* o salida.

Cada neurona esta conectada con otras a través de enlaces, formando así capas de neuronas que operan en paralelo y propagan la señal o información de una capa a la siguiente, manteniendo únicamente una dirección de transmisión de información, es decir, de atrás hacia delante, hecho por el cual a estas redes se les denomina Redes Feedforward.

2.1.1. Perceptrón Simple

En el año 1958 Frank Rosenblatt [8] crearía un algoritmo de reconocimiento de patrones basado en una red de aprendizaje de computador de dos capas. A este algoritmo Rosenblatt lo llamaría Perceptrón.

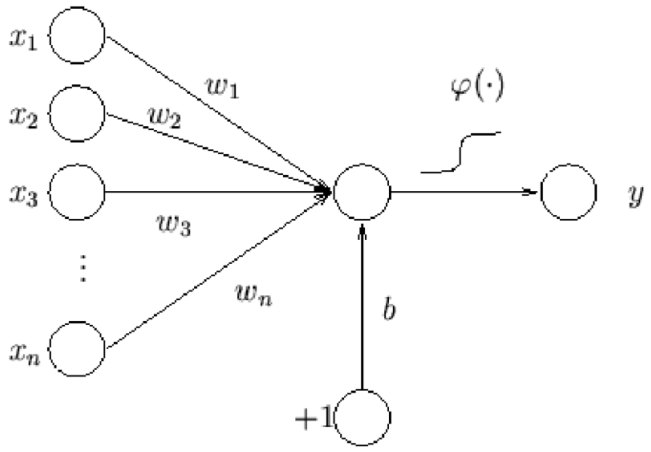


Figura 2.1: Diagrama de un Perceptrón.

Como podemos observar en la figura 2.1, el Perceptrón es la unidad básica de una red, la cual recibe una serie de entradas $x_1..x_n$, que a su vez, se combinan con una serie de pesos ponderados $w_1..w_n$. Dependiendo de si resultado de la combinación sobrepasa cierto umbral, el Perceptrón devolverá una salida u otra. A esta salida se le aplica una función de activación Φ, la cual dependiendo del tipo de función aplicada, puede devolver un valor u otro (explicadas en la subsección 2.1.3).

Además, dado que la derivada se utilizará más tarde en el entrenamiento de la propia red neuronal, es conveniente que dicha derivada sea posible expresarla en términos de la función original.

Cada neurona se puede expresar como la siguiente ecuación matemática:

$$y = \Phi(s) = \Phi(\Sigma_{i=1}^{n} w_i x_i + b) \tag{2.1}$$

El algoritmo del Perceptrón simple ajusta los pesos de manera proporcional a la diferencia existente entre la salida actual de la red y la salida deseada, con el objetivo de minimizar el error actual de la red neuronal.

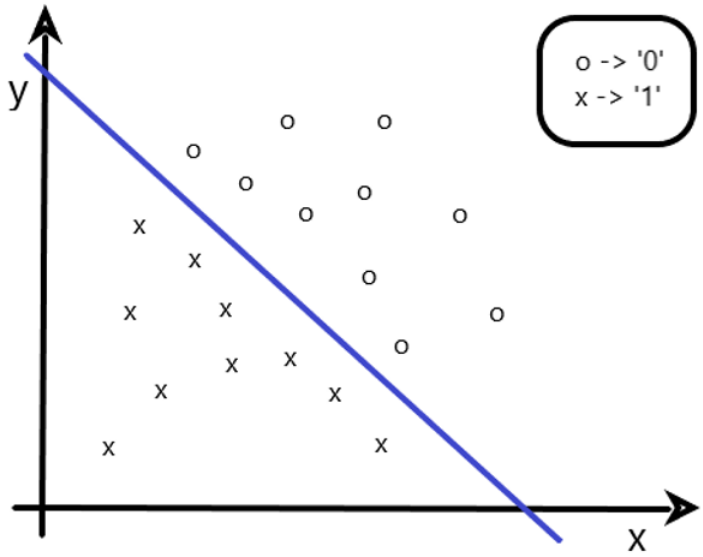


Figura 2.2: Región de decisión correspondiente a un Perceptrón simple con dos neuronas de entrada.

Rosenblatt demostró que el algoritmo siempre converge en un tiempo finito y con independencia de los pesos de partida siempre que la función a representar sea linealmente separable. Como podemos observar en la Figura 2.2, el algoritmo de entrenamiento del Perceptrón consigue clasificar todas las muestras en dos regiones o clases, estableciendo una frontera de decisión.

2.1.2. Perceptrón Multicapa (MLP)

Tal y como acabamos de ver, el Perceptrón simple tiene una limitación, y es que solo puede discriminar patrones que sean separables por un hiperplano, en el caso de la Figura 2.2, una recta, ya que solo disponemos de dos neuronas.

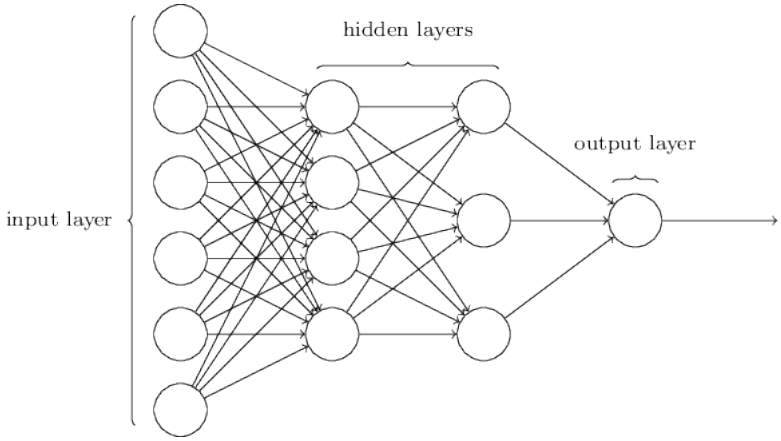


Figura 2.3: Diagrama de un Perceptrón Multicapa totalmente conectado.

La combinación de varios perceptrones simples en más de una capa, resuelve el problema anterior, ya que esta combinación hace posible la resolución de problemas no lineales.

Denominamos 'capa' al conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior.

En 1986 Rumelhart y otros autores presentaron la *Regla Delta Generalizada* o algoritmo de *Back Propagation (BP)*[9] para adaptar los pesos propagando los errores hacia las capas ocultas de la red. Es de esta forma que se suele entrenar el perceptrón multicapa o MLP(de sus siglas en inglés, *Multilayer Perceptrón*), de ahí que a este tipo de redes se les conozca con el sobrenombre de redes de retropropagación.

Los MLP cuentan con tres tipos distintos de neuronas tal y como podemos ver en la Figura 2.3:

- Las neuronas de la capa de entrada, las cuales únicamente reciben las muestras sin modificar.

- Las neuronas de las capas ocultas, (denominadas de esta forma porque el usuario únicamente visualiza la entrada y salida de la red), las cuales se encargan de realizar el cómputo de la propia red neuronal.

- Las neuronas de la capa de salida, las cuales se encargan de realizar la clasificación en las diferentes clases gracias a la función de activación que usualmente es distinta a la utilizada en las capas ocultas.

Tal y como demostró Hornik [10] en 1991, un Perceptrón Multicapa con al menos una capa oculta y que disponga de una función de activación la cual no sea lineal, puede aproximar relaciones no lineales entre los datos de entrada y salida. Por lo tanto, el MLP, funciona como un aproximador universal de funciones y por lo tanto, pudiendo definir cualquier función.

2.1.3. Funciones de Activación

Como hemos dicho anteriormente, la función de activación es la encargada de devolver una salida a partir de una entrada, por lo que determinará como será la salida final de una neurona. Cabe decir, que es importante que las derivadas de las funciones sean simples ya que de esta forma, minimizaremos el coste computacional.

A continuación presentaremos las funciones de activación más utilizadas.

Sigmoide: la función sigmoide (ecuación 2.2), transforma los valores introducidos a una escala [0,1], donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de igual forma a 0. Algunas de sus características es que tiene una convergencia lenta y se suele utilizar en la última capa en problemas de clasificación binaria.

$$f(x) = \frac{1}{1 - e^{-x}} \tag{2.2}$$

ReLU: la función *Rectified Lineal Unit* O ReLU (ecuación 2.3), anula los valores negativos dejando los positivos como tal. Esta función no está acotada y además se ha demostrado que se comporta bien en cualquier tipo de tarea.

$$f(x) = max(0, x) \tag{2.3}$$

Softmax: esta función (ecuación 2.4) , transforma las salidas en una representación de probabilidades, de tal forma que el sumatorio de todas las probabilidades de las salidas es 1. Esta función al representar probabilidades, esta acotada entre 0 y 1. Además, Softmax es algo diferente a las anteriores ya que en este caso, se tienen en cuenta todas las neuronas de la capa a la vez, en lugar de una sola neurona. Se suele emplear en lugar de la sigmoide para problemas de clasificación de dos o más clases.

$$f(x)_j = \frac{e^{z_j}}{\Sigma_{k=1}^{K} e^{z_k}} \tag{2.4}$$

Sobre cual es la mejor función a aplicar en cada neurona, esta es una pregunta compleja, ya que no existe una base de conocimientos teóricos que certifiquen que la utilización de una sea mejor que otra. Pese a lo anterior, se ha demostrado empíricamente que la función ReLU es la que mejor funciona en las capas ocultas (motivo por el cual se utiliza en el presente proyecto). Por otra parte, la elección de funciones de activación, es una rama de investigación muy activa y en desarrollo.

2.1.4. Técnicas de regularización.

A la hora de entrenar una red neuronal artificial, esta puede llegar al caso de 'memorizar' los datos de entrenamiento y por tanto caer en el sobre-entrenamiento. Esto supone un problema ya que, con los datos de entrenamiento, obtendremos unos resultados excelentes, pero cuando le introduzcamos nuevos datos a clasificar a la red, nos daremos cuenta que los resultados son mucho peor de lo esperado.

Esto es debido a que los modelos que son entrenados con un gran porcentaje de datos del corpus los cuales tienen poca varianza, tienden a memorizar y por tanto les cuesta generalizar. Con el objetivo de evitar lo anterior y que nuestro modelo no sobre-entrene y por lo tanto memorice los datos de entrenamiento, debemos aplicar estas técnicas.

En nuestro caso hemos utilizado principalmente, la denominada Normalización por lotes (*Batch Normalization*) [20] pese a que existen otras como son *Dropout* (explorado su uso en la subsección 5.2.1 - Exploración de hiper-parámetros) [21], decaimiento de los pesos (*Weight decay)* o aumento de los datos (*Data agumentation*).

Batch Normalization consiste en añadir un paso entre las neuronas y la función de activación con el objetivo de normalizar las activaciones de salida. Dicha técnica reduce la variación de la co-varianza entre las mustras y, además, normaliza los valores de la post-activación haciendo que estos no sean ni muy altos ni muy bajos. Esta técnica es aplicada sobre cada conjunto de datos pequeño o (*mini-batch*) en lo que es dividido el conglomerado de los datos de entrenamiento.

A la normalización anteriormente descrita se le añaden dos parámetros, el *bias* y una constante por la que multiplicaremos en cada activación. Esto ayudará a nuestro modelo a la hora de ajustar los datos de entrada, además de reducir las oscilaciones de la función de coste. Por lo tanto podremos aumentar el *learning rate* (o tasa de aprendizaje) ya que la red tenderá a converger hacia un mínimo global sin riesgo de caer en un mínimo local.

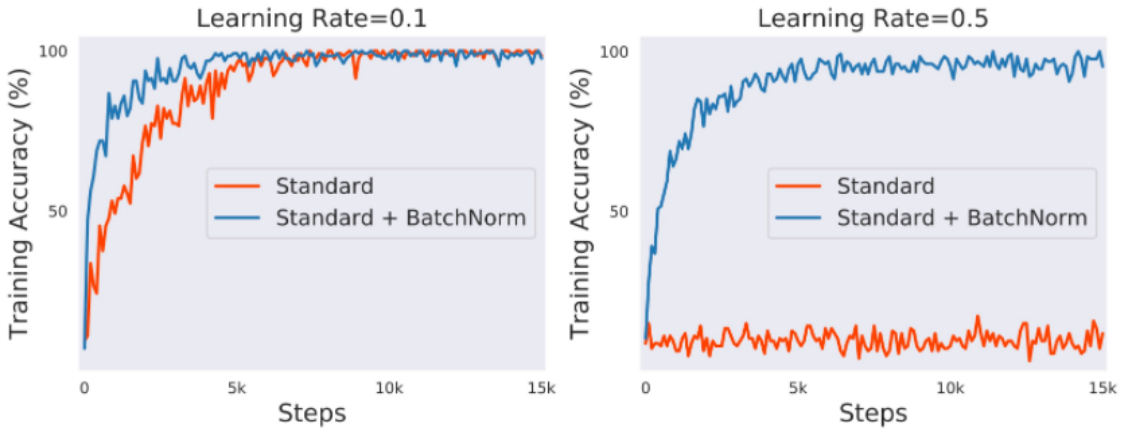


Figura 2.4: Comparación del uso de Batch Normalization. Figura extraída de [22]

En la figura 2.4 podemos ver como, con el uso de Batch Normalization, la red converge antes que sin el uso de la misma, y que además podemos aumentar el *learning rate* convergiendo aun así, al contrario que es el caso sin técnica de regularización.

Por otro lado, *Dropout* es una técnica de regularización que consiste en eliminar o desactivar de forma aleatoria con cierta probabilidad, que es previamente definida, algunas neuronas de las capas ocultas del modelo. Esto permite tener diferentes modelos para alcanzar una mejor predicción evitando además el sobre-entrenamiento.

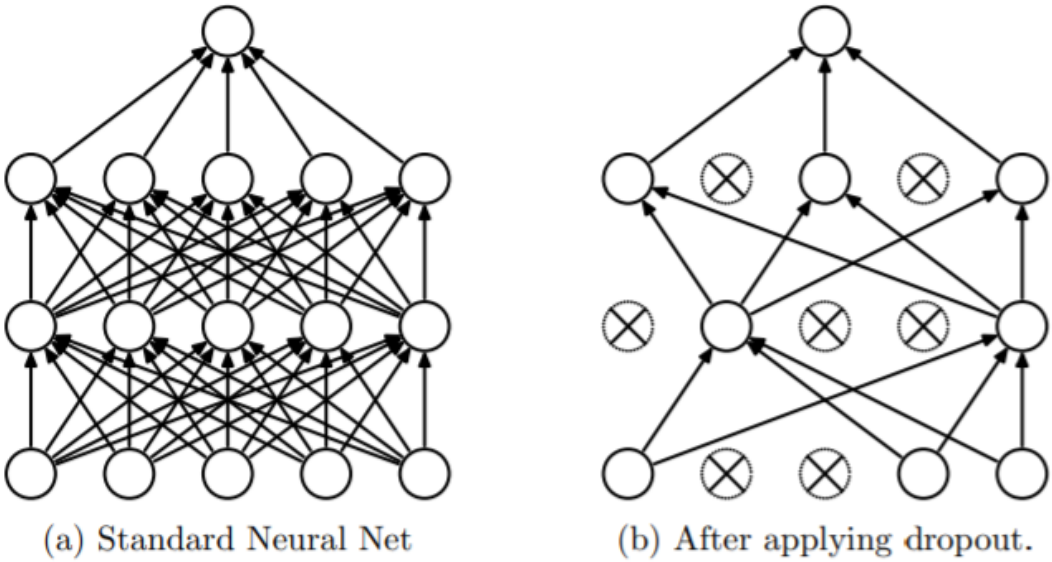


Figura 2.5: Comparación del uso de Dropout. Figura extraída de [21]

En la parte b de la figura 2.5, observamos como se desactiva cierto número de neuronas creando así un modelo diferente y evitando que ninguna neurona *'memorice'* parte de los datos, evitando el sobreajuste de la red.

Hay que indicar, que es posible emplear *Batch-Normalization* y *Dropout* a la vez, pero realmente no aporta nada nuevo, ya que las dos buscan los mismos objetivos. Además el incluir *Dropout* en un modelo con *Batch-Normalization* podría ralentizar el proceso de aprendizaje, sin mejorar los resultados.

2.1.5. Optimizadores.

Los optimizadores son piezas clave en el entrenamiento de redes neuronales. Estos se encargan de acelerar el entrenamiento de nuestra red, cambiando parámetros de la misma como los pesos o el *learning rate* con el objetivo de reducir las perdidas.

Al igual que pasaba con las técnicas de regularización, podemos encontrar muchos optimizadores, los cuales emplearan diferentes métodos y algoritmos para acelerar nuestro modelo. Algunos de los más utilizados son:

SGD: El SGD (por sus siglas en inglés: Stochastic Gradient Descendent) [23], selecciona las muestras de entrenamiento de manera aleatoria, escogiendo únicamente una muestra al mismo tiempo para actualizar los parámetros de la red.

Algunas de las ventajas de utilizar SGD son la actualización frecuente de los parámetros del modelo, lo que le hace converger en poco tiempo, y no requiere gran cantidad de memoria, ya que no guarda los valores de las funciones de perdida.

Por otro lado, las desventajas de emplear SGD son, la gran varianza entre los parámetros de la red, y que, comparado con los enfoques más nuevos, tiene más probabilidad de quedarse atascado en un mínimo local.

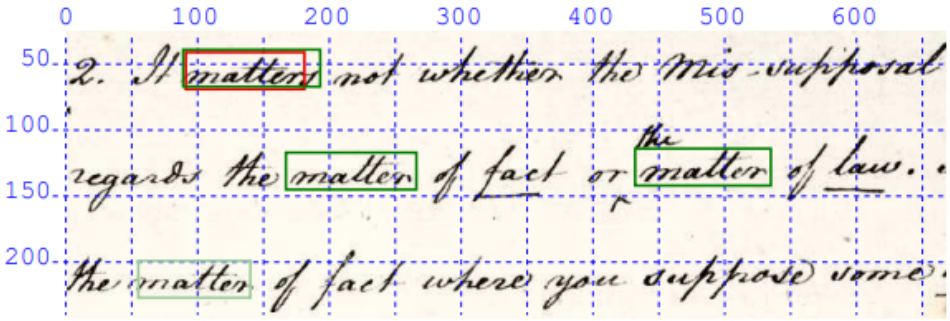
RMSprop: El optimizador RMSprop (del inglés: Root Mean Square Prop) [26] es una extensión del anterior. La principal diferencia con SGD es que, mientras en este mantenemos una única tasa de aprendizaje, en RMSprop esta tasa se adapta cambiando con el tiempo.

Adam: El optimizador Adam (por sus siglas en inglés: Adaptive Moment Estimation) [24] combina todo lo bueno de RMSprop y otro optimizador denominado AdaGrad [**?**]. En este se emplea una tasa de aprendizaje variable con el tiempo que además se ve afectado por la media del gradiente de un parámetro denominado *momentum*.

Algunas de las ventajas de usar Adam como optimizador, es que es muy eficiente computacionalmente y requiere pocos requisitos de memoria.

2.2 Índices Probabilísticos

Los índices probabilísticos consisten en una lista de palabras obtenidas de una imagen, indicando las probabilidades de encontrar cada palabra, además de las coordenadas donde se encuentra dicha palabra.




```
# pageID="Bentham-071-021-002-part"
#    keyword relPrb    bounding box
#
          2  0.929    1  36  20 31
         21  0.064    1  36  24 31
         IT  0.982   33  36  27 31
         IF  0.012   33  36  26 31
    MATTERS  0.998   76  35 104 31
     MATTER  0.011   77  36  93 31
        NOT  0.999  216  36   7 31
    WHETHER  1.000  256  36  99 31
        THE  0.997  389  36  33 31
MIS-SUPPOSAL 1.000  455  36 193 31

        THE  0.927  430  88  30 31
        LHE  0.056  434  88  25 31
        ...  ...      ...     ...
```

```
REGARDS  0.857    5 115  84 31
UGARDS   0.138    5 115  80 31
    THE  0.993  110 115  43 31
 MATTER  0.998  160 115  93 31
     OF  0.996  271 115  23 31
   FACT  0.999  306 115  49 31
     OR  0.973  377 115  37 31
     ON  0.021  377 115  42 31
 MATTER  0.990  425 116 100 31
     OF  0.995  542 115  25 31
    LAM  0.407  575 115  30 31
   BIMR  0.175  575 115  55 31
    ...  ...      ...     ...
    LAW  0.032  575 115  36 31
   TAUE  0.031  575 115  55 31
    ...  ...      ...     ...
   LANE  0.012  575 115  59 31
```

```
    THE  0.990    1 198  28 31
 MATTER  0.934   61 198  64 31
     OF  0.988  141 198  28 31
   FAST  0.367  182 198  62 31
    FAR  0.186  182 198  36 31
    ...  ...      ...     ...
   FACT  0.017  182 198  46 31
     AS  0.142  200 198  29 31
    HAE  0.022  200 198  29 31
  WHERE  0.992  255 198  90 31
    YOU  0.761  365 198  45 31
    YOW  0.030  365 198  45 31
   GOUS  0.064  372 198  47 31
SUPPOSE  0.975  429 198 120 31
SUPFROSE 0.024  429 198 125 31
   SOME  0.834  570 198  78 31
 SONER   0.016  576 198  83 31
    OME  0.109  580 198  65 31
     ME  0.022  620 198  22 31
```

Figura 2.6: Ejemplo de índice probabilístico. Figura extraída de [11].

Para el cálculo de los dichos índices probabilísticos, se ha calculado la probabilidad a posteriori de la palabra de la siguiente forma:

$$P(v|X,i,j), 1 \leq i \leq I, 1 \leq j \leq J, v \epsilon V \tag{2.5}$$

Donde X es una imagen de texto digital con dimensiones I x J, V es el vocabulario e *(i,j)* son píxeles de X. Por lo tanto, P(*v* | X,*i,j*) es la probabilidad de que una palabra *v* este escrita en una región de X que incluya al píxel *(i,j)*. A la región que contiene ese píxel, se le denomina *bounding box*.

Para el cálculo de esta probabilidad, se considera que cada palabra *v*, esta escrita en todos los posibles *bounding boxes* de X, con la ecuación siguiente:

$$P(v|X,i,j) = \sum_{b \epsilon B(i,j)} P(v,b|X,i,j) = \sum_{b \epsilon B(i,j)} P(b|X,i,j) P(v|X,b,i,j) \tag{2.6}$$

Donde *B(i,j)* es el conjunto de todos los *bounding boxes* de la imagen que contengan el pixel *(i,j)*.

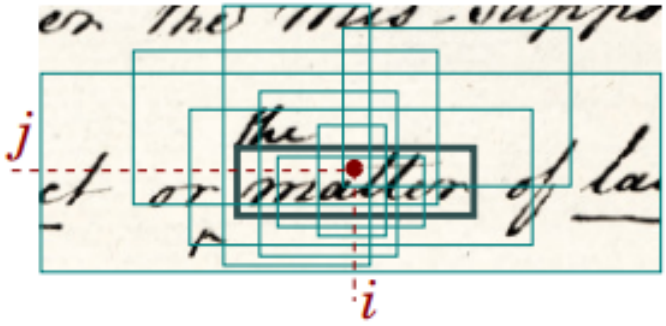


Figura 2.7: *Bounding boxes b* ϵ B(*i,j*). Para *v* = "matter". Figura extraída de [11].

En la figura 2.7 podemos observar como se lleva a cabo el cálculo anteriormente explicado. Vemos que la línea más gruesa indica mayor probabilidad de P(*v* | X,b) mientras que las lineas más finas u otros *bounding boxes* contribuyen a la suma.

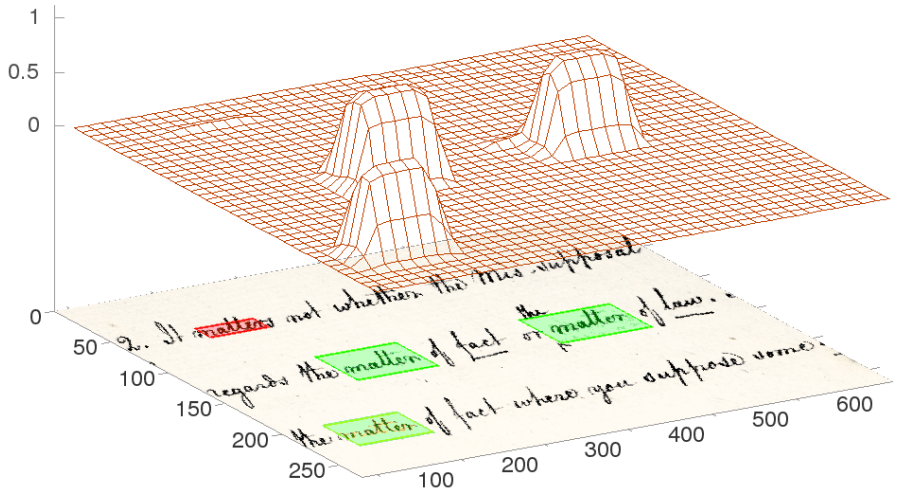


Figura 2.8: Ejemplo de calculo de probabilidad a posteriori para *v* = "matter". Figura extraída de [11].

Posteriormente a la extracción de los índices probabilísticos es posible realizar una búsqueda de palabras en los documentos manuscritos con una confianza o umbral específico. Dicha búsqueda es conocida como *Keyword spotting*.

2.3 Information Gain (IG)

Para la predicción de la clase del documento, no todas las palabras ayudan de la misma forma. Por esto es útil realizar una ordenación de las mismas.

Una buena forma de realizar esta depuración es calculando el IG de cada palabra de nuestro vocabulario V y quedándonos, únicamente, con las n palabras que mayor IG posean. Creando así un vocabulario V_n<V.

Por tanto, entendiendo que t_v es un valor booleano que será verdadero si, para cualquier documento de la colección, aparece la palabra v en este. P(t_v) como la probabilidad de que algún documento contenga la palabra v y P($\overline{t_v}$) como la probabilidad de que ningún documento contenga v.

El IG de una palabra v está definido como:

$$\begin{aligned} IG(v) = &- \sum_{c \epsilon C} P(c) \log P(c) + \\ &P(t_v) \sum_{c \epsilon C} P(c|t_v) \log P(c|t_v) + \\ &P(\overline{t_v}) \sum_{c \epsilon C} P(c|\overline{t_v}) \log P(c|\overline{t_v}) \end{aligned} \tag{2.7}$$

Donde P(c) es la probabilidad a priori de la clase c, P(c | t_v) la probabilidad condicional de que un documento pertenezca a la clase c, dado que este contiene la palabra v, y P($c|\overline{t_v}$), la probabilidad de que un documento pertenezca a la clase c y no contenga v.

Con el objetivo de estimar las probabilidades anteriormente expuestas, serán definidas de la siguiente forma:

$$P(t_v) = \frac{f(t_v)}{M} \tag{2.8}$$

$$P(\overline{t_v}) = \frac{M - f(t_v)}{M} \tag{2.9}$$

$$P(c|t_v) = \frac{f(c, t_v)}{f(t_v)} \tag{2.10}$$

$$P(c|\overline{t_v}) = \frac{M_c - f(c, t_v)}{M - f(t_v)} \tag{2.11}$$

Donde, $f(t_v)$ será el numero de documentos de la colección que contienen la palabra v, y f ($\overline{t_v}$) será el numero de documentos que no contiene v.

M_c son los documentos de la colección pertenecientes a la clase c y $f(c, t_v)$ el número de esos documentos que contienen v.

Una vez hecho el calculo del Information Gain para V, este puede ser ordenado de forma decreciente. De esta forma crearíamos un ranking sobre las palabras que más información aportan sobre la clase de documento en la que estas se encuentran.

Posteriormente, podremos usar esta lista escogiendo las n primeras palabras para entrenar nuestro clasificador.

2.4 Tf·Idf (Term frequency - Inverse document frequency)

El Tf·Idf (del inglés Term frequency - Inverse document frequency), es una técnica que nos da la relevancia para una palabra $v \epsilon$ V de un documento d en una colección D (d ϵ D). Es decir nos ayuda a encontrar las palabras más relevantes para un documento en concreto.

El valor de Tf·Idf aumenta proporcionalmente con el número de veces que aparece v en el documento, pero esto es compensado con la frecuencia de la palabra en la colección de documentos. De modo que, Tf(v/d) es un estimador de la probabilidad condicional de una palabra v, dado un documento d, es decir P(v/d) y por lo tanto, Tf quedaría definido de la siguiente forma:

$$Tf = \frac{f(v,d)}{f(d)} \tag{2.12}$$

Donde *f (v,d)* es el número de veces que la palabra v aparece en el documento d, y *f (d)* es el número total de palabras que encontramos en el documento.

Por otro lado, Idf(v), refleja la importancia del término o palabra v en un la colección de documentos. Hay que remarcar, que la tarea que realizamos con Idf, sería posible ejecutarla también con IG, pero numerosos estudios [13][14][15] argumentan que es preferible la aplicación de Idf. Por consecuente, podemos definir Idf como:

$$Idf = \frac{log(M)}{f(t_{\mathrm{v}})} \tag{2.13}$$

Poniéndolo todo junto, para representar un documento d como un vector de características $\vec{d}$, el valor de cada característica, d_v, es calculado como el Tf·Idf de d y v:

$$\mathrm{d_v} = \mathrm{Tf} \cdot \mathrm{Idf}(v, d) = \mathrm{Tf}\ (v,d) \cdot \mathrm{Idf}\ (v)$$

$$= P(v|d) \log \frac{1}{P(t_v)} = \frac{f(v,d)}{f(d)} \log \frac{M}{f(t_v)} \tag{2.14}$$

2.5 Análisis de Texto Utilizando Indexación Probabilistica

En esta sección analizaremos como, gracias a la indexación probabilística, podemos estimar características básicas del texto, como puede ser el numero de palabras que contiene un documento o el número de documentos que contienen una palabra. Características que han sido explicadas en las secciones anteriores 2.3 y 2.4 para la estimación de probabilidades.

La indexación probabilística o PRIX (de sus siglas en inglés *Probabilistic Indexing*) fue propuesto para mitigar la incertidumbre intrínseca que encontramos a nivel de palabra en los textos manuscritos, especialmente, si estos son históricos y antiguos. Es por esto que la principal característica por la cual utilizamos PRIX, es por que nos permite realizar una rápida y precisa búsqueda en textos manuscritos como los que tratamos en este trabajo.

Esta técnica de indexación ha sido probada con anterioridad para indexar largas colecciones de manuscritos antiguos como los de la cancillería medieval francesa [16], los Bentham Papers [17] y la colección del Archivo General de las Indias (AGI) [7].

PRIX se basa en conceptos e ideas como *keyword spotting*, excepto que en lugar de almacenar únicamente las palabras clave o *keywords*, se detecta y almacena cualquier elemento de una imagen que pueda ser interpretado como una palabra con su probabilidad de relevancia o RP (por sus siglas en inglés *Relevance Probability*) y su localización en la imagen. A estos elementos se les denomina *pseudo-words.*

La indexación probabilística resuelve, por tanto, un problema de clasificación binaria donde se decide si una palabra v está o no escrita en una región x de una imagen. Dicho de otra forma, debemos de averiguar si x es relevante dado v esa '*relevancia*' es a la que llamamos RP.

Es por tanto que probabilidad de relevancia puede ser estimada a partir de la siguiente ecuación:

$$P(R|x,v) = \sum_{b \subseteq x} P(R,b|x,v) \approx max_{b \subseteq x} P(v|x,b) \tag{2.15}$$

Donde, R es una variable aleatoria binaria, b es un *Bounding Box (BB)*, x es una región particular de una imagen, y por tanto con $b \subseteq x$ queremos indicar que b es un BB dentro del conjunto de BBs que encontraríamos en x.

La probabilidad de relevancia P(R | x,v) puede considerarse como la esperanza estadística de que v se encuentre en la región x. Por lo tanto, la suma de todos los RPs para todas las *pseudo-words* indexadas en x debería de aproximarse al número exacto de palabras escritas en x, es decir *n(x)*.

El valor esperado de *n(x)* puede ser estimado con la siguiente ecuación:

$$E[n(x)] = \sum_{v} P(R|x,v) \tag{2.16}$$

La ecuación anterior puede ser fácilmente extendida para estimar el número total de palabras o *running words* de un documento completo X. Por consiguiente, el número de palabras esperado en X, *n(X)*, puede ser estimado como:

$$E[n(X)] = \sum_{x \subseteq X} \sum_{v} P(R|x,v) \tag{2.17}$$

Por otra parte, la frecuencia de una *pseudo-word* específica v en un documento X, *n(v,X)*, puede ser estimado como:

$$E[n(v,X)] = \sum_{x \subseteq X} P(R|x,v) \tag{2.18}$$

Otra de las frecuencias útiles en la clasificación de documentos y que hemos utilizado en la sección previa para el cálculo de IG y Tf·Idf (secciones 2.3 y 2.4), es el número de documentos de la colección, $\mathcal{X}$, que contienen un palabra v:

$$E[m(v,\mathcal{X})] = \sum_{X \subseteq \mathcal{X}} max_{x \epsilon X} P(R|x,v) \tag{2.19}$$

CAPÍTULO 3
Trabajo Realizado

En esta sección de la memoria explicaremos que técnicas se han seguido para la implementación de las distintas herramientas para la clasificación de los documentos manuscritos según su contenido.

Primeramente, se explicará cómo, a partir de los índices probabilísticos, se han estimado tanto IG como Tf·Idf. Posteriormente, se expondrá que tipo de red neuronal artificial ha sido implementada en este proyecto al igual que las características de la red.

Cabe recordar, que todo el código desarrollado durante el proyecto, se puede encontrar en el repositorio público [1].

3.1 Estimación de IG y Tf·Idf a partir de PRIX

Information Gain (IG) y Tf·Idf son técnicas las cuales, como hemos visto en sus secciones correspondientes (secciones 2.3 y 2.4), se basan en frecuencias. Esto puede ya sea en frecuencias de palabras por documento o por documento y clase, o frecuencias de los documentos que contienen una palabra específica.

Pero, para una colección de imágenes de texto no transcritas, no es posible el cálculo de estas frecuencias directamente. Es por esto que, como hemos explicado en la sección 2.5, estas frecuencias se pueden estimar a partir de los índices probabilísticos de cada imagen de texto.

Por lo tanto, de acuerdo a la notación escrita en las sección 2.5, un documento D de las secciones 2.3 y 2.4, es en este caso es un conjunto de imágenes, es decir, X. Además, un conjunto de documentos, $\mathcal{D}$, pasa a ser una colección de imágenes $\mathcal{X}$ donde definimos $\mathcal{X}_c$ como la colección de imágenes de documentos de una clase determinada, c. Así pues, M es el número total de documentos es decir, M = $|\mathcal{X}|$ y M_c = $|\mathcal{X}_c|$ es el número total de documentos pertenecientes a la clase c.

Asimismo, las frecuencias necesarias para calcular el IG de una palabra v han sido estimadas de la siguiente forma a partir de los índices probabilísticos de $\mathcal{X}$:

El número de documentos que contienen v:

$$f(t_v) = \sum_{X \subseteq \mathcal{X}} max_{x \epsilon X} P(R|x, v) \tag{3.1}$$

El número de documentos de la clase c que contiene v:

$$f(c, t_v) = \sum_{X \subseteq \mathcal{X}_c} max_{x \epsilon X} P(R|x, v) \tag{3.2}$$

[1] Link del repositorio: https://github.com/JuanjoFlores/TFG-ETSINF

Por otra parte, las frecuencias necesarias para el cálculo de Tf·Idf, además de f(t_v), han sido estimadas de la siguiente forma a partir de los índices probabilísticos:

El número total de palabras que aparecen en un documento D:

$$f(D) = \sum_{x \subseteq X} \sum_{v} P(R|x, v) \tag{3.3}$$

El número de veces que aparece una palabra en un documento D:

$$f(v, D) = \sum_{x \subseteq X} P(R|x, v) \tag{3.4}$$

Teniendo en cuenta lo anterior, estas han sido las ecuaciones que se han implementadas mediante Python 3.9, creando así un script para el cálculo de IG y otro para el de Tf·Idf. Entre ambos se crea una matriz la cual utilizaremos para el entrenamiento de la red.

En la matriz, encontramos una primera columna con los nombres de los documentos seguido de los valores de Tf·Idf para cada palabra de la primera fila. Cabe decir que el orden de las palabras no es aleatorio, si no que están ordenadas según su IG. Esto nos será de mucha utilidad a la hora de entrenar el modelo o red neuronal artificial implementada, ya que a la hora de escoger un número específico de palabras, siempre escogeremos las que mayor IG tengan.

Por último, es importante señalar, que la estructura de datos que ha sido utilizada para la estimación de las distintas frecuencias, son diccionarios Python. Esto es debido a su fácil uso tanto en el almacenamiento de los datos, como en su extracción. Además los diccionarios Python, poseen un gran número de operaciones y métodos facilitando así su manipulación.

Por otra parte, hay que añadir, que el hecho que la complejidad algorítmica en el acceso a los datos sea del O(n) en el peor de los casos y O(1) la complejidad promedio, hace que sea positivo para el proyecto el uso de esta estructura de datos.

3.2 Red Neuronal Artificial

Una vez que hemos estimado tanto el IG como el TF·Idf, y por tanto, tenemos la representación de las imágenes de los documentos, es momento de implementar la red neuronal. El objetivo de esta es la estimación de la clase c ϵ C a la que pertenece un documento d ϵ D. Esta predicción se basa en el marco estadístico de riesgo de error mínimo:

$$c^*(X) = argmax_{c\epsilon\{1,..,C\}} P(c|X) \tag{3.5}$$

Donde X es una representación vectorial, resultado del cálculo del Tf·Idf, y P(c | X) es la probabilidad a posteriori, la cual puede ser calculada siguiendo varios enfoques. En este trabajo, se seguirá el del Perceptrón Multicapa (MLP).

La arquitectura que ha sido implementada para la estimación de la probabilidad a posteriori anteriormente citada, ha sido la siguiente:

```
    super(Net, self).__init__()
    self.layers = layers
    #Input Layer
    model = [
        nn.Linear(len_feats,layers[0]),
        nn.BatchNorm1d(layers[0]),
        nn.ReLU()
    ]
    #Hidden Layers
    for i, _ in enumerate(layers[:-1]):
        model += [
            nn.Linear(layers[i],layers[i+1]),
            nn.BatchNorm1d(layers[i+1]),
            nn.ReLU()
        ]
    #Output Layer
    model += [
        nn.Linear(layers[-1], n_classes)
    ]
    self.num_params = 0
    self.model = nn.Sequential(*model)
    for param in self.model.parameters():
        self.num_params += param.numel()

def forward(self, inp):

    res = self.model(inp)
    return  F.log_softmax(res, dim =-1)
```

Figura 3.1: Arquitectura implementada.

Esta arquitectura está implementada mediante Pytorch[27] y se puede ver parte del modelo en la figura 3.1. En dicha arquitectura dependiendo del número de capas (layers) que se le pase a la función, efectuaremos el Perceptrón Simple (explicado en la sección 2.1.1) o el Perceptrón Multicapa (explicado en la sección 2.1.2).

En esta arquitectura, en el caso de emplear capas ocultas en la red, aplicamos *Batch Normalization* (explicado el la sección 2.1.4) tanto en la capa inicial como en las ocultas, utilizado para acelerar y estabilizar la red mediante la normalización de las capas donde se aplica. Además, en la salida de las capas ocultas, como función de activación, empleamos ReLU (explicada en la sección 2.1.3).

En cuanto a la función de activación en la capa de salida, emplearemos softmax (explicada en la sección 2.1.3) para que, utilizando todas las características extraídas a partir de las capas anteriores, calcule la probabilidad de que el documento d pertenezca la clase $c \epsilon C$.

En el proyecto han sido considerados tres variantes de MLP: el MLP-1, el MLP-2 y el MLP-3, donde el número después del guión indica el número de capas ocultas que el modelo posee. Estas variantes han sido escogidas siguiendo la metodología explicada en [7]. Además de las tres variantes anteriores también realizaremos experimentos con otros MLP con el objetivo de comprobar si obtenemos mejores resultados que con las anteriores variantes o no.

Para concluir, en cuanto a el número de neuronas por capa, pese a que no existe un método para determinar el número óptimo, tras realizar experimentos con 128, 256 y 512, decidimos quedarnos con 256 ya que obteníamos mejores resultados que con los otros dos valores como número de neuronas. Por lo tanto, las diferentes variantes de MLP llevarán 256 neuronas por capa.

CAPÍTULO 4
Corpus

En este capítulo se explicará el corpus utilizado para este proyecto, así como su procedencia y utilidad.

4.1 Archivo Histórico Provincial de Cádiz s.XVII

Figura 4.1: Ejemplo de páginas del corpus del legajo JMDB_4949.

El Archivo Provincial de Cádiz es creado el 12 de noviembre de 1931 con el fin de recoger y custodiar toda la documentación notarial centenaria que no estuviera custodiada por los Colegios Notariales.

Su principal función es la de conservar el patrimonio documental provincial y prestar servicio a los investigadores para su uso y consulta.

Los archivos notariales sobre los cuales tratamos en el presente proyecto, están compuestos por 16849 libros o protocolos, con 800 páginas de media manuscritas, las cuales contienen unos 250 actos notariales de naturaleza diferente (poder, arrendamiento, testamento...).

Debido a la falta de personal, este fondo documental solo cuenta con escuetos índices como instrumentos de descripción, en los que, para cada localidad, se especifica el número concreto de notarías, los nombres de sus titulares a lo largo del tiempo y las fechas extremas de actividad.

Es por tanto, que se carecen de instrumentos más precisos y detallados (catálogos) que permitan a los investigadores la identificación, localización y consulta de sus fondos mediante búsquedas simples o complejas de diferente naturaleza (toponímicas, onomásticas, temáticas). Por lo tanto, la única alternativa que tienen los usuarios es la de enfrentarse a leer una a una las páginas de los expedientes o protocolos, hasta encontrar la información de interés.

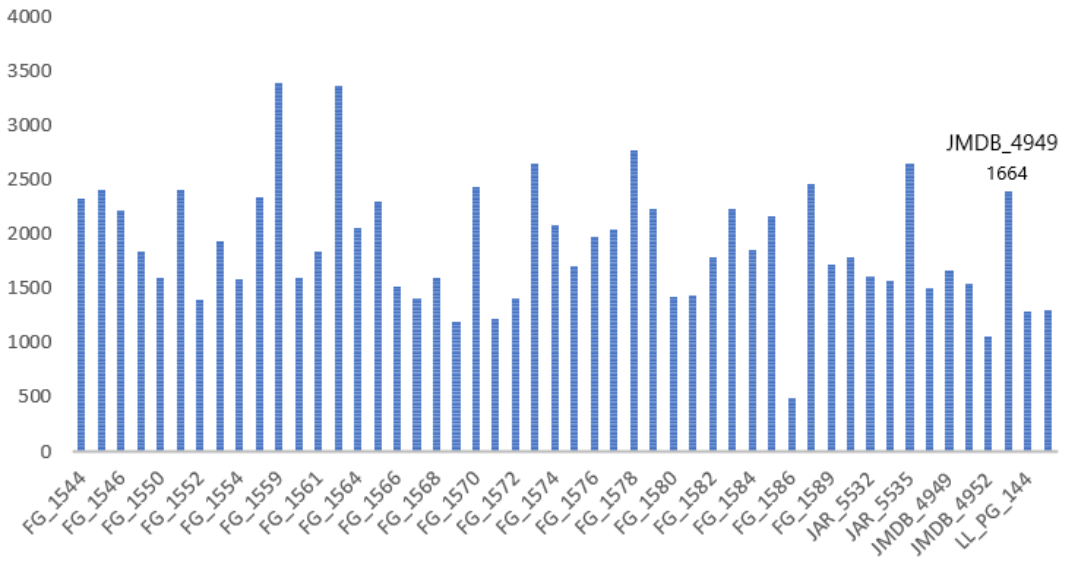


Figura 4.2: Distribución del número de páginas por legajo.

En la figura 4.2 podemos observar la distribución del número de páginas por legajo pertenecientes al Archivo Provincial de Cádiz. Entre estos, se encuentra el JMDB_4949, el cual, será empleado para los experimentos en este proyecto como corpus. Este legajo, pertenece al fondo del Distrito de Cádiz y corresponde al año 1723.

El legajo JMDB_4949 ha sido el seleccionado para este proyecto debido a que los índices probabilísticos extraídos del legajo en cuestión, poseen unas características idóneas para la clasificación por clase de los expedientes.

Además de lo anterior, el legajo JMDB_4949, también ha sido escogido debido a su tamaño, ya que, no posee un tamaño muy grande ni muy pequeño, por lo que es apropiado para comenzar con la clasificación de expedientes por clase de cada legajo.

En la figura 4.3 contemplamos la distribución que sigue el número de páginas por expediente del legajo JMDB_4949.

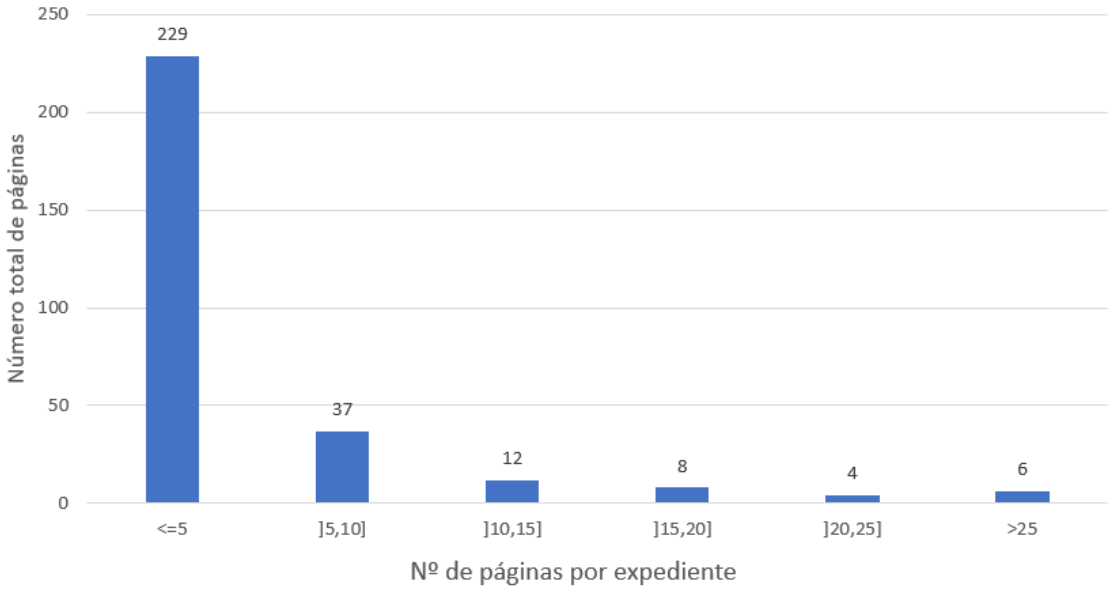


Figura 4.3: Histograma del nº de páginas por expediente perteneciente al legajo JMDB_4949.

En este histograma vemos que la mayoría de los expedientes únicamente ocupan 5 o menos páginas, estos, al ser mayoría tendrán una mayor influencia en los resultados finales. Es por esto que para una mejor interpretación de los resultados, hemos creído necesario realizar el siguiente histograma:

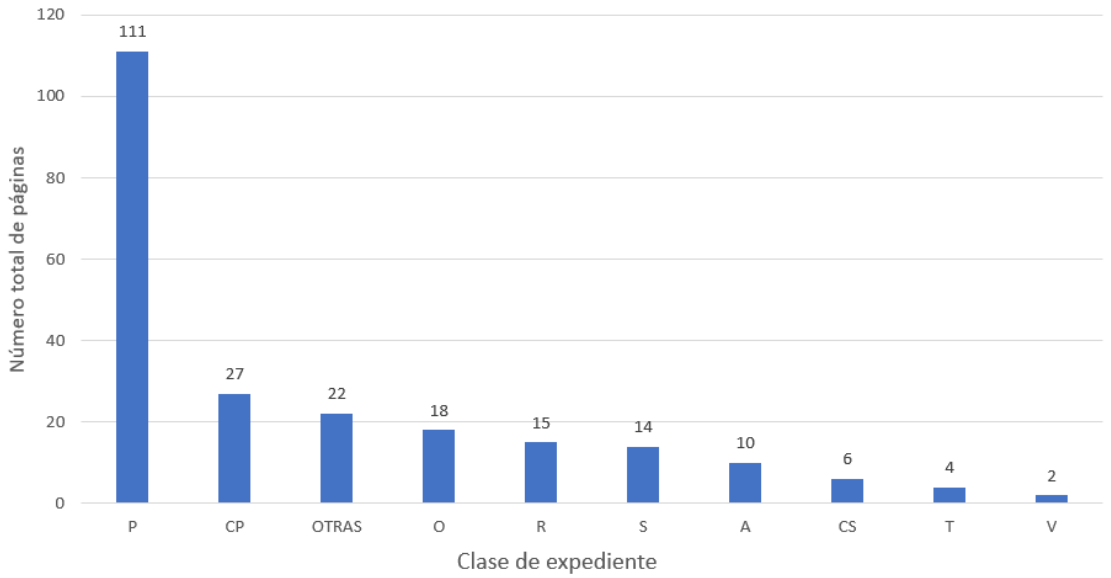


Figura 4.4: Histograma de nº de páginas por clase para expedientes de 5 o menos páginas.

En la figura 4.4 podemos observar que, el 49 % de los expedientes cuya longitud es menor o igual que 5, pertenecen a la clase P, que como veremos en la tabla 4.1, es la clase Poder. También vemos una clase denominada 'Otras', esto es debido a que, como explicaremos posteriormente, no todas las clases se han tenido en cuenta para la clasificación.

Cabe destacar, que las primeras 50 páginas del protocolo o legajo, pertenecen al índice anteriormente descrito que poseen todos los expedientes, por lo tanto, dichas 50 páginas no se han tenido en cuenta para los experimentos llevados a cabo en este proyecto.

En la figura 4.4 encontramos un ejemplo de página perteneciente al legajo JMDB_4949. En esta imagen se puede observar en el lado superior izquierdo la frase: *Venta de Casas*, por lo que este expediente casi con toda seguridad pertenece a la clase de Venta y por lo tanto, debería de ser clasificado como tal.

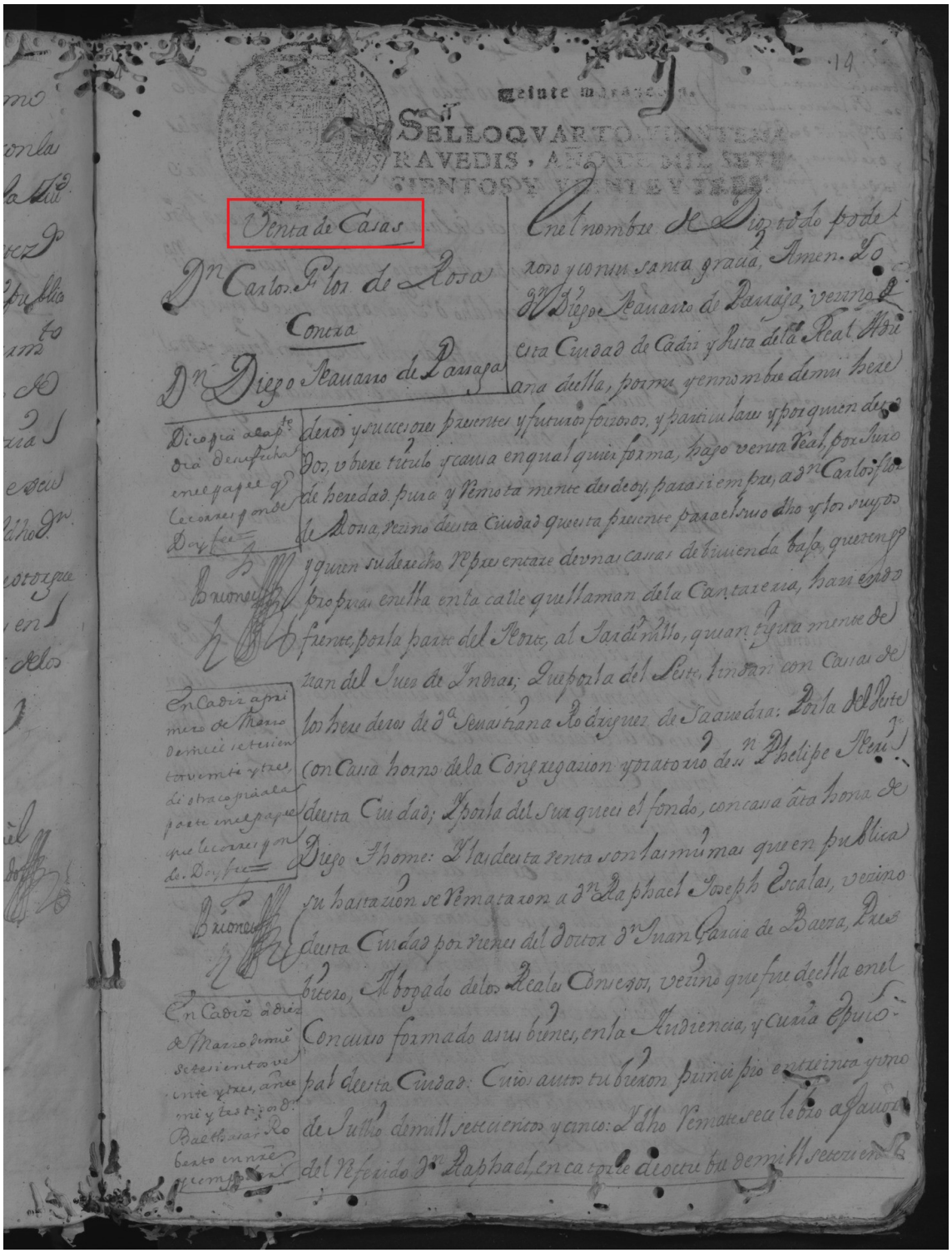


Figura 4.5: Ejemplo de página del corpus del legajo JMDB_4949 pertenecientes a la clase V (Venta).

En la figura 4.6, podemos observar otro ejemplo de una página procedente del legajo JMDB_4949, pero que esta vez, pertenece a un expediente de la clase Carta de Pago, ya que podemos observar la frase: *carta de pago* en la imagen (recuadro rojo).

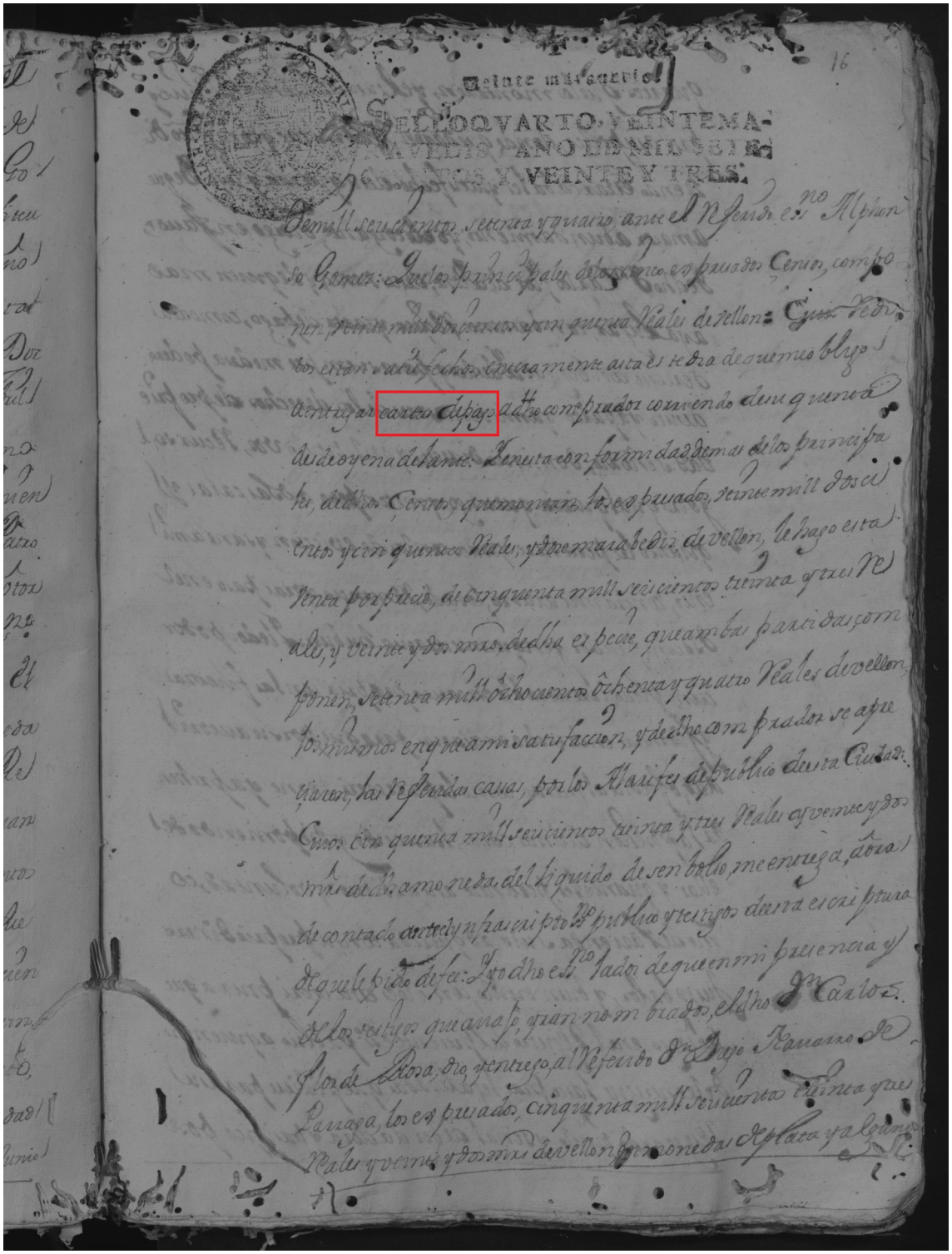


Figura 4.6: Página del corpus del legajo JMDB_4949 perteneciente a la clase CP (Carta de Pago).

Tal y como hemos explicado anteriormente, los expedientes pertenecen a más de 20 clases diferentes dependiendo de su contenido:

Clase	Tipología	Muestras	Nº de Páginas	Media Pags./Exp.
P	Poder	127	436	3.43
CP	Carta De Pago	35	158	4.50
O	Obligación	21	90	4.28
S	Sustitución	15	80	5.33
R	Riesgo	15	60	4.00
V	Venta	14	314	22.43
CS	Censo	12	134	11.17
A	Arrendamiento	12	56	4.66
T	Testamento	11	66	6.00
D	Declaración	5		
DP	Depósito	4		
TH	Tratado de hecho	3		
N	Nombramiento	2		
LI	Libertad esclavo	2		
DT	Dote	2		
AP	Aprovisionamiento	2		
TRAS	Traspaso	1		
RES	Redención de censo	1		
RED	Redención	1		
REC	Reclamación	1		
LI	Licencia	1		
INF	Información	1		
F	Fianza	1		

Tabla 4.1: Tabla con las distintas clases que podemos encontrar en el legajo JMDB_4949 ordenadas por número de muestras.

Como podemos observar, la tabla se encuentra ordenada por el número de muestras que encontramos en la colección de documentos. Para este proyecto, se ha decidido tener en cuenta únicamente las nueve primeras clases debido a que no disponemos de las muestras necesarias en algunos casos.

Además, también podemos encontrar el número de páginas totales de las que disponemos de cada clase, así como la media del número de páginas por expediente.

CAPÍTULO 5
Experimentos y resultados

En este capítulo se mostrará y explicará los experimentos que han sido llevados a cabo en este proyecto y sus detalles, así como sus resultados.

5.1 Detalles de los experimentos

Comenzaremos con los detalles que hemos tenido en cuenta en la elaboración del módulo de cálculo de IG y Tf·Idf.

En estos dos módulos, previo a la estimación de probabilidades, se ha realizado una poda de la probabilidad de las palabras de los índices probabilísticos. Esta poda consiste en escoger únicamente pseudo-palabras cuya probabilidad de aparición en el documento sea superior a un umbral de 0.1.

Para escoger dicho umbral, previamente se han hecho experimentos con distintos valores: 0.5, 0.1 y 0.01, obteniendo los mejores resultados con 0.1. Debido a que, en algunos casos se desperdiciaba información posiblemente útil de pseudo-palabras con menor probabilidad y en otro muchas de ellas no aportaban demasiado o se correspondían con palabras mal escritas o errores.

Por lo que concierne a los detalles del entrenamiento de las diferentes variantes de MLP, se han inicializado con los siguientes parámetros siguiendo [19] con: *max-epoch* = 100, *num features* (en nuestro caso, número de palabras seleccionadas) en potencias de dos, desde 8 a 16384 y un *batch size* = 4.

Además, los modelos han sido entrenados y evaluados con una partición de entrenamiento y una de evaluación, además calculando la precisión media y con *cross-entropy loss* de 100 *epochs* utilizando el optimizador SGD (*Stochastic Gradient Descendent*) para actualizar los pesos de la red y con una tasa de aprendizaje del 0.01.

Respecto a la tasa de aprendizaje, se tomó la decisión de utilizar 0.1 después de probar con 0.01, 0.1, 1 y 10, obteniendo los mejores resultados en los experimentos con la tasa o *learning rate* escogido.

En lo que concierne al optimizador, se ha tomado la decisión de utilizar SGD, posteriormente de compararlo con Adam (Adaptative Moment Estimation) [24] ya que era el que se empleaba en [7] y comprobar que con el primero obteníamos mejores resultados que con el segundo.

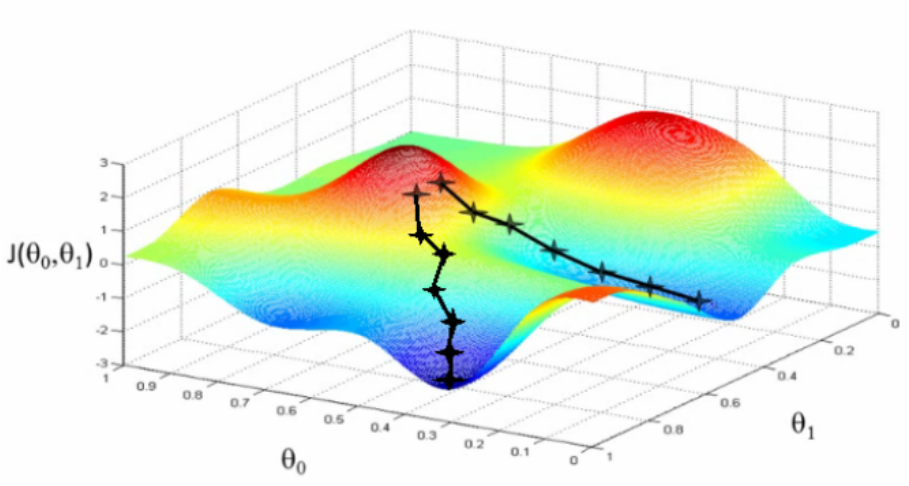


Figura 5.1: Ejemplo del funcionamiento del algoritmo de SGD (Stochastic Gradient Descendent).

En la figura 5.1 podemos observar el funcionamiento del algoritmo SGD, en esta podemos ver, que para dos inicializaciones del algoritmo para la misma función, este encuentra dos mínimos diferentes. Es por lo anterior, que en el trabajo se han hecho un total de 100 inicializaciones quedándonos finalmente con la mejor.

También observamos los 'saltos' (cruces negras) que da el algoritmo hasta encontrar los mínimos. La distancia que observamos entre estos 'saltos' dependerá de la tasa de aprendizaje o *learning rate* que estemos utilizando.

Para evitar los efectos de inicialización aleatorios, y a raíz de lo anteriormente dicho, con el objetivo de obtener resultados más consistentes, cada experimento, se ha ejecutado un total de 100 veces con diferentes semillas de inicialización y calculado el promedio de los resultados de precisión o *accuracy*.

Para el cálculo de las matrices de confusión que veremos en la sección de resultados, se ha calculado de la misma forma la media de las 100 inicializaciones, obteniendo así la matriz de confusión promedio.

5.2 Resultados

En esta sección, finalmente, exploraremos detenidamente los diferentes resultados obtenidos de los experimentos llevados a cabo con las tres variantes de MLP. Posteriormente, realizaremos una exploración de hiper-parámetros en los que veremos los resultados de las distintas modificaciones.

En cuanto a los resultados de entrenar los tres modelos MLP, estos pueden ser vistos en la siguiente figura:

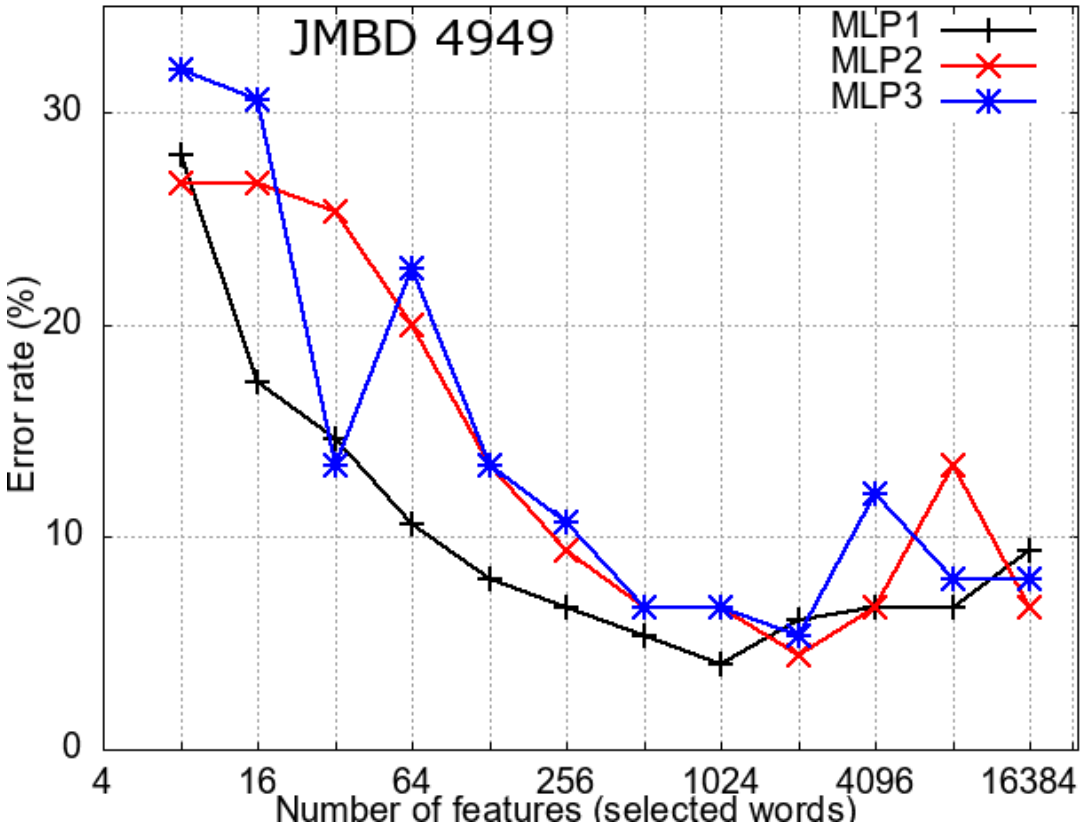


Figura 5.2: Error de clasificación para las tres variantes de MLP con 256 neuronas por capa, un *batch-size* de 4 y empleando SGD como optimizador. Los intervalos de confianza son del 95 % (no mostrados para mayor claridad), todos menores del ±10,0 %.

En la figura 5.2, observamos las distintas curvas de las tasas de error para las tres variantes de MLP, donde cada resultado es la media para 100 inicializaciones tal y como hemos explicado anteriormente.

En la tabla siguiente podemos ver los valores reales que hemos obtenido para las distintas inicializaciones (no se muestran todos los resultados para una mejor exposición de los mismos):

Error (%) - MLP						
Nº Palabras	MLP-1		MLP-2		MLP-3	
8	28.00	[18,39]	26.66	[17,37]	32.00	[22,43]
16	17.33	[9,26]	26.66	[17,37]	30.60	[21,42]
32	14.60	[7,23]	25.33	[16,36]	13.33	[6,22]
64	10.66	[4,18]	20.00	[11,30]	22.66	[14,33]
128	8.00	[2,15]	13.33	[6,22]	13.33	[6,22]
256	6.66	[2,13]	9.33	[3,16]	10.66	[4,18]
512	5.33	[1,11]	6.66	[2,13]	6.66	[2,13]
1024	4.00	[0,9]	6.66	[2,13]	6.66	[2,13]
2048	8.00	[2,15]	4.00	[0,9]	5.33	[1,11]
4096	6.66	[2,13]	6.66	[2,13]	12.00	[5,20]
8192	6.66	[2,13]	13.33	[6,22]	8.00	[2,15]
16384	9.33	[3,16]	6.66	[2,13]	8.00	[2,15]

Tabla 5.1: Tabla con los porcentajes de error e intervalo de confianza para la clasificación con las diferentes modelos MLP con 256 neuronas por capa, un *batch-size* de 4 y empleando SGD como optimizador.

A partir de la tabla anterior y la figura 5.2, podemos comprobar que el mejor resultado ha sido tanto con la variante MLP-1 como con MLP-2. Con estas, se llega a conseguir un error del 4 % y 4.4 % respectivamente, es decir, un acierto del 96 % y 95.6 % , al seleccionar un total de 1024 palabras para realizar el entrenamiento del modelo MLP-1 y 2048 para MLP-2.

También comprobamos que, la variante MLP-1 se comporta de una manera más *'suave'*, ya que, como vemos, en su curva no hay picos demasiado acentuados como si los hay en las otras dos. Es por esto que en caso de empate en cuanto a mejor resultado entre MLP-1 y MLP-2, como ha ocurrido en este caso, sería más aconsejable elegir la primera.

En lo referente al intervalo de confianza empleado (no se muestra en la figura para mayor claridad), este es del 95 %, es por esto que en el mejor resultado (4 %), su media intervalo de confianza referente al error es: [0 %,9 %].

El intervalo de confianza describe la variabilidad entre la medida obtenida, en nuestro caso 4 %, y el valor real. Este rango se corresponde con una distribución normal entre la cual se encuentra con una alta probabilidad el valor real. Esta 'alta probabilidad' en nuestro caso es del 95 %.

Centrándonos ya en la mejor configuración posible, es decir, MLP-1 con 1024 palabras, encontramos la siguiente matriz de confusión:

P	CP	O	S	R	V	CS	A	T	
38	0	0	0	0	0	0	0	0	P
0	3	0	0	0	0	0	0	0	CP
0	0	10	0	0	0	0	0	0	O
0	0	1	3	0	0	0	0	0	S
1	0	0	0	3	0	0	0	0	R
1	0	0	0	0	5	0	0	0	V
0	0	0	0	0	0	3	0	0	CS
0	0	0	0	0	0	0	4	0	A
0	0	0	0	0	0	0	0	3	T

En la anterior matriz, cada columna representa el número de predicciones de cada clase mientras que cada fila representa a las instancias en la clase real. Es por lo tanto que la matriz de confusión nos ayuda para comprobar en que clases de ha equivocado el modelo en la predicción de la misma. De esta forma podemos comprobar el desempeño de nuestro modelo.

A partir de lo anterior, podemos comprobar que el modelo ha errado un total de 3 veces. Este número se corresponde correctamente con el porcentaje anteriormente contemplado en la gráfica (4 %), ya que si dividimos el número total de muestras, 75, entre el número de fallos, 3, obtenemos 0.04, es decir 4 %.

Los fallos se corresponden con las clases S (Sustitución), R (Riesgo) y V (Venta). Donde, para la clase S asigna de forma errónea una muestra a la clase O (Obligación), y, por otro lado, para las clases R y V, asigna de forma equívoca una muestra a la clase P (Poder).

Estos errores pueden deberse a diversos factores tales como la falta de muestras de esa clase o a que en la muestra mal clasificada aparecen palabras representativas de otra clase que no es la correcta.

Por otro lado, también observamos que los documentos pertenecientes a la clase P (Poder), los cuales poseían un tamaño de 5 o menos páginas (Figura 4.4), el modelo predice de manera correcta la clase de todas las muestras.

5.2.1. Exploración de hiper-parámetros

En esta sección adjuntaremos los resultados obtenidos en la exploración de los diferentes hiper-parámetros de los modelos anteriormente expuestos.

Comenzaremos con la variación del nº de neuronas en la arquitectura MLP-1 para los diferentes nº de palabras. Es por tanto que, exploraremos MLP-1 con 8, 16 y 32 neuronas, con el objetivo de comprobar si se obtienen mejores resultados. Para todas los experimentos, al igual que antes, los resultados serán la media de 100 inicializaciones y se utilizará un *learning rate* = 0.01.

Los resultados obtenidos son los siguientes:

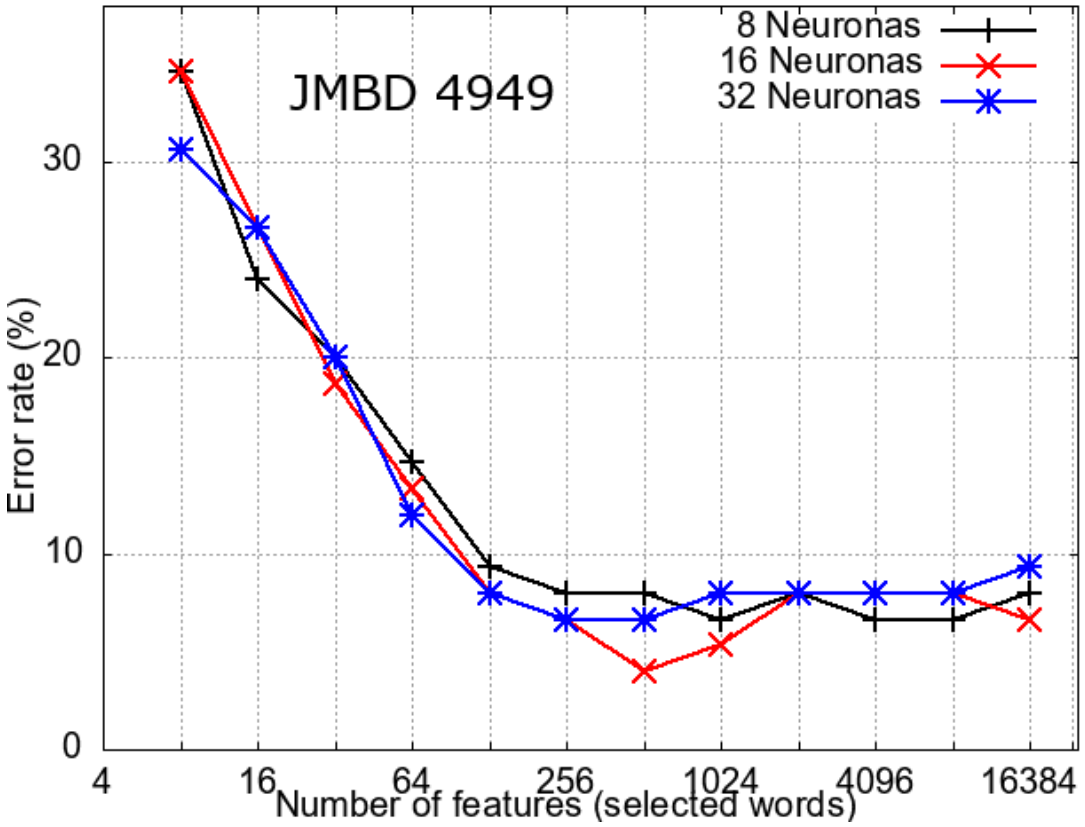


Figura 5.3: Error de clasificación para las tres variantes de MLP-1 con 8,16 y 32 neuronas respectivamente, un *batch-size* de 4 y empleando SGD como optimizador. Los intervalos de confianza son del 95 % (no mostrados para mayor claridad), todos menores del ±10,0 %.

A la vista de la figura podemos extraer que, el cambio en el Nº de neuronas, no ha producido una mejora notoria en los resultados. Debido a esto, si comparamos las curvas obtenidas en la figura 5.3 con la obtenida en la 5.2 no encontraremos mejora, incluso en los casos de 8 y 32, se empeora el resultado, ya que no consigue obtener ese error mínimo del 4 % al que habíamos llegado anteriormente y que si que obtenemos con 16 neuronas por capa.

Error (%) - Nº de Neuronas						
Nº Palabras	8		16		32	
8	34.66	[24,46]	34.66	[24,46]	30.60	[21,42]
16	24.00	[15,34]	26.66	[17,37]	26.66	[17,37]
32	20.00	[11,29]	18.66	[10,28]	20.00	[11,29]
64	14.66	[7,23]	13.33	[6,22]	12.00	[5,20]
128	9.33	[3,16]	8.00	[2,15]	8.00	[2,15]
256	8.00	[2,15]	6.66	[2,13]	6.66	[2,13]
512	8.00	[2,15]	4.00	[0,9]	6.66	[2,13]
1024	6.66	[2,13]	5.33	[1,11]	8.00	[2,15]
2048	8.00	[2,15]	8.00	[2,15]	8.00	[2,15]
4096	6.66	[2,13]	8.00	[2,15]	8.00	[2,15]
8192	6.66	[2,13]	8.00	[2,15]	8.00	[2,15]
16384	8.00	[2,15]	6.66	[2,13]	9.33	[3,16]

Tabla 5.2: Tabla con los porcentajes de error e intervalo de confianza para la clasificación con las diferentes modelos MLP con 8,16 y 32 neuronas respectivamente, un *batch-size* de 4 y empleando SGD como optimizador.

Es por lo anterior y a la vista de la tabla 5.2, que aunque si que observamos mayor *'fluidez'* en las curvas, no hemos conseguido mejorar los resultados ya obtenidos en la sección anterior, por lo que, en este caso, la exploración del Nº de neuronas no afecta de gran manera al resultado final obtenido, y es por esto que no hemos continuado explorando otras posibilidades.

Continuando con la exploración de hiper-parámetros, llega el turno de la experimentación con la variación del *batch-size*. Como hemos dicho en la sección 5.1 Detalles de los experimentos, el *batch-size* con el cual hemos ejecutado todos los experimentos era de 4, a continuación contemplaremos los resultados de variar este hiper-parámetro del modelo.

Los experimentos han sido llevados a cabo con MLP-1 con 256 neuronas, un *learning rate* de 0.01 y un *batch-size* de 16, 32 y 64. A continuación podemos observar los resultados obtenidos en la siguiente figura:

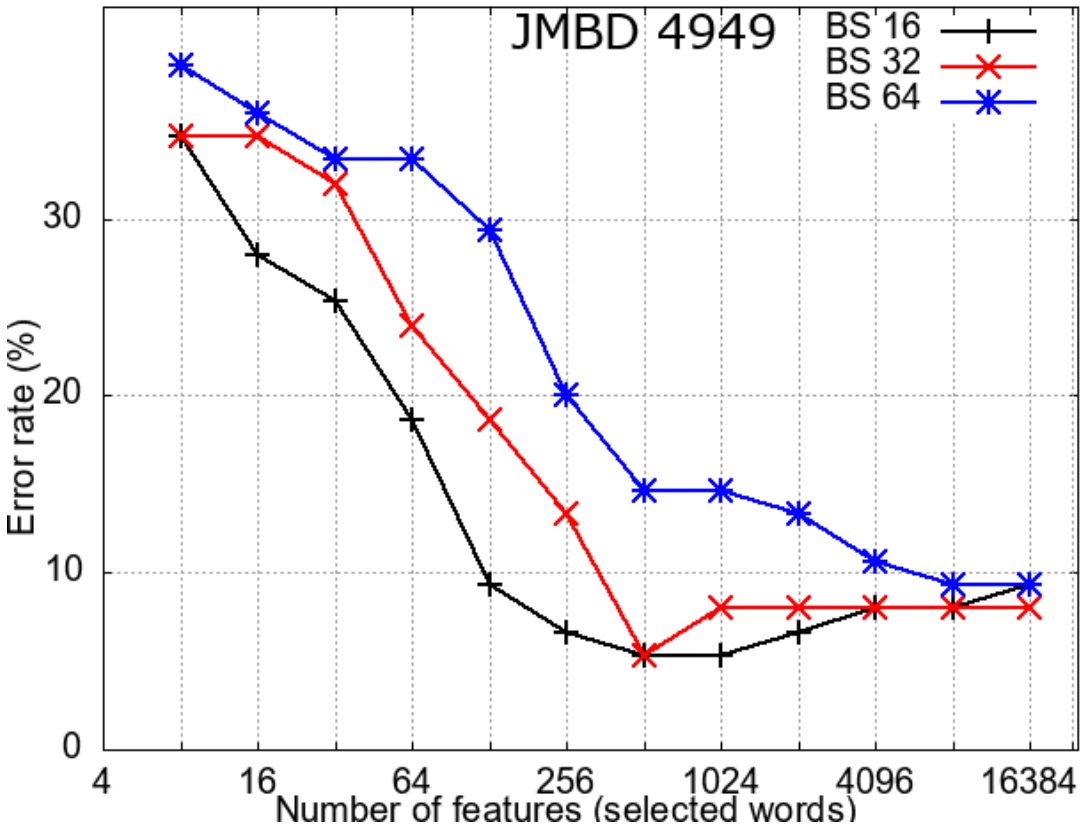


Figura 5.4: Error de clasificación para las tres variantes de MLP-1 con 256 neuronas, un *batch-size* de 16, 32 y 64 respectivamente y empleando SGD como optimizador. Los intervalos de confianza son del 95 % (no mostrados para mayor claridad), todos menores del $\pm$10,0 %.

En vista de los resultados obtenidos, podemos asegurar que no es conveniente utilizar un *Batch-size* demasiado alto, ya que, como vemos en la Figura 5.4, los peores resultados se obtienen con el *Batch-size* más alto, mientras que los mejores son obtenidos con 16, es decir, el menor.

Es por lo anterior, que no hemos continuado explorando mayores números de *Batch-size*, ya que podemos ver una tendencia a empeorar los resultados.

En la siguiente tabla podemos ver todos los resultados obtenidos:

Error (%) - Batch-size						
Nº Palabras	16		32		64	
8	34.66	[24,46]	34.66	[24,46]	38.66	[26,50]
16	28.00	[18,39]	34.66	[24,46]	36.00	[26,46]
32	25.33	[16,26]	32.00	[23,42]	33.30	[23,44]
64	18.66	[10,28]	24.00	[15,34]	33.30	[23,44]
128	9.33	[3,16]	18.66	[10,28]	29.33	[19,39]
256	6.66	[2,13]	13.33	[6,12]	20.00	[11,29]
512	5.33	[1,11]	5.33	[1,11]	14.66	[7,23]
1024	5.33	[1,11]	8.00	[2,15]	14.66	[7,23]
2048	6.66	[2,13]	8.00	[2,15]	13.33	[6,22]
4096	8.00	[2,15]	8.00	[2,15]	10.66	[4,18]
8192	8.00	[2,15]	8.00	[2,15]	9.33	[3,16]
16384	9.33	[3,16]	8.00	[2,15]	9.33	[3,16]

Tabla 5.3: Tabla con los porcentajes de error e intervalo de confianza para la clasificación con modelos MLP con 256 neuronas, un *batch-size* de 16, 32 y 64 respectivamente y empleando SGD como optimizador.

A partir de los datos expuestos en la tabla 5.3, podemos concluir que, aunque obtenemos resultados nada despreciables con un *Batch-size* de 16 y 32, llegando a un error del 5.33 %, no llegamos a igualar los resultados previamente logrados con el *Batch-size* de 4, con el que obteníamos un error del 4 %.

A continuación, siguiendo con la exploración de hiper-parámetros, es el momento de variar los optimizadores, ya que, durante todo el proyecto hemos empleado SGD y nos parece interesante conocer que resultados obtendríamos en el caso de emplear Adam y RMSprop.

Los experimentos por lo tanto, han sido ejecutados con MLP-1 con 256 neuronas, un *learning-rate* = 0.01, un *batch-size* = 4 y variando el optimizador utilizado. Los resultados obtenidos son los siguientes:

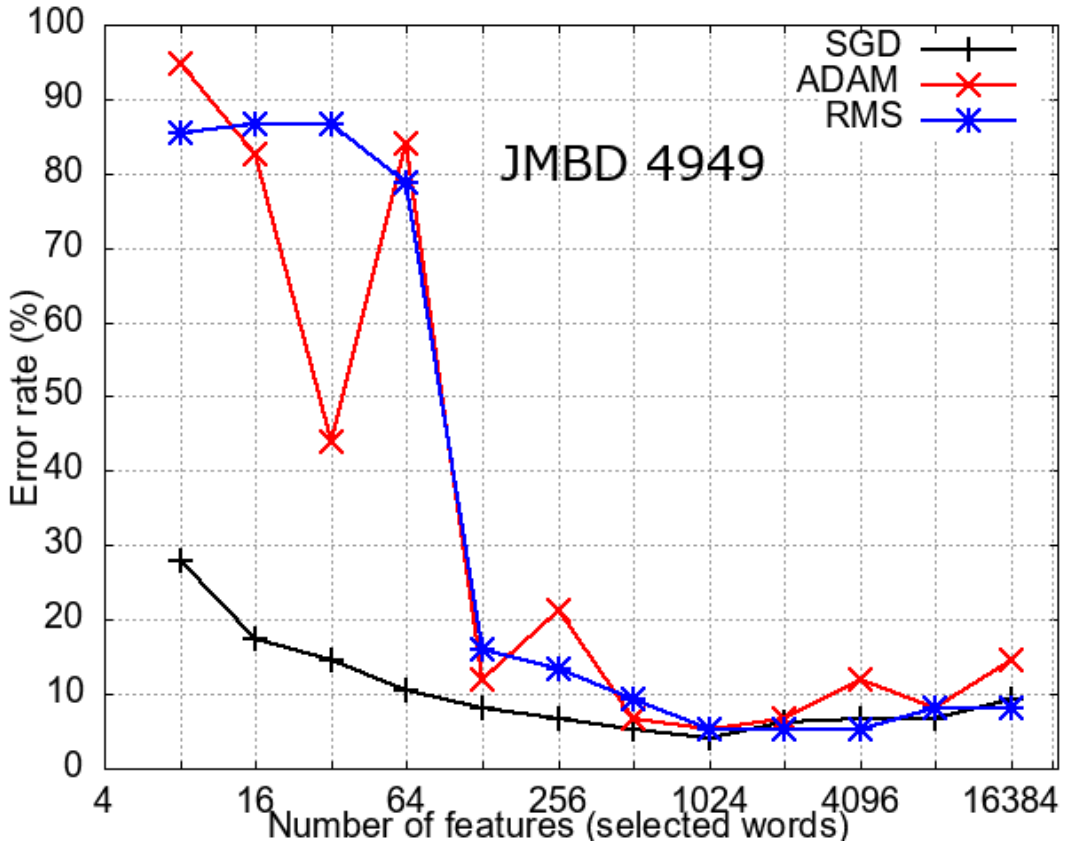


Figura 5.5: Error de clasificación para las tres variantes de MLP-1 con 256 neuronas, un *batch-size* de 4 y empleando SGD, ADAM y RMSprop como optimizador. Los intervalos de confianza son del 95 % (no mostrados para mayor claridad), todos menores del $\pm$10,0 %.

En la figura 5.5, podemos observar el comportamiento de los diferentes optimizadores para cada número de palabras seleccionadas. A la vista de los resultados, podemos concluir que el que mejor responde para ante cualquier número de palabras es SGD, ya que podemos ver que su curva es más suave que la de los otros dos.

Por otro lado, comprobamos que tanto Adam como RMSprop, obtienen unos resultados decentes a partir de emplear más de 128 palabras, por lo que no sería nada aconsejable emplear estos en un modelo con pocas características de entrada.

Los resultados obtenidos pueden ser visualizados en la siguiente tabla:

Error (%) - Optimizador						
Nº Palabras	SGD		ADAM		RMSprop	
8	28.00	[18,39]	94.66	[0,90]	85.33	[78,94]
16	17.33	[9,26]	82.66	[75,92]	86.66	[89,95]
32	14.60	[7,23]	44.00	[33,56]	86.66	[89,95]
64	10.66	[4,18]	84.00	[15,34]	78.66	[70,88]
128	8.00	[2,15]	12.00	[5,20]	16.00	[8,25]
256	6.66	[2,13]	21.33	[13,31]	13.33	[6,22]
512	5.33	[1,11]	6.66	[2,13]	9.33	[3,16]
1024	4.00	[0,9]	5.33	[1,11]	5.33	[1,11]
2048	8.00	[2,15]	6.66	[2,13]	5.33	[1,11]
4096	6.66	[2,13]	12.00	[5,20]	5.33	[1,11]
8192	6.66	[2,13]	8.00	[2,15]	8.00	[2,15]
16384	9.33	[3,16]	14.66	[7,23]	8.00	[2,15]

Tabla 5.4: Tabla con los porcentajes de error e intervalo de confianza para la clasificación con modelos MLP con 256 neuronas, un *batch-size* de 4 y empleando SGD, ADAM y RMSprop como optimizador

A partir de los datos de la tabla 5.4, podemos confirmar que ADAM y RMSprop obtienen unos resultados pésimos con un número de características bajo, mientras que SGD se mantiene más estable. Además, encontramos que el error más bajo, y por lo tanto el mejor resultado lo encontramos en la columna de SGD con 4.00 %, ya que con ADAM y RMSprop, lo mejor que obtenemos es un error del 5.33 %.

Por último, también comprobamos que, dependiendo el número de características, tal vez nos sea más conveniente utilizar un optimizador u otro, tal y como observamos en la tabla, y no emplear para todas el mismo optimizador.

CAPÍTULO 6
Conclusiones

El objetivo principal que nos impusimos al comienzo de este proyecto era el de afianzar la reciente tecnología de los índices probabilísticos en el campo del Machine Learning y de la clasificación y reconocimiento de texto manuscrito. Así como la exploración del legajo JMDB_4949 procedente del Archivo Provincial de Cádiz con lo que aportar resultados al proyecto Carabela.

Para ello, se trataría de emplear las técnicas utilizadas en [7] para la clasificación de expedientes del legajo JMDB_4949, en las diferentes clases. Además de explorar diferentes hiper-parámetros con el objetivo de encontrar nuevos resultados.

En cuanto a la meta principal, podemos concluir que hemos presentado un modelo el cual es capaz de realizar una clasificación de documentos basada en el contenido obteniendo unos resultados bastante reseñables, llegando a obtener un error del 4 % con la variante MLP-1.

Para esto, hemos empleado técnicas tradicionales para la clasificación de textos manuscritos, estimando las frecuencias de palabras necesarias a partir de índices probabilísticos. De este modo, exploramos la reciente técnica de los índices probabilísticos, a la par que, se supera la necesidad de transcribir explícitamente los documentos manuscritos, tarea la cual es inviable cuando se trata de grandes colecciones como la tratada en el presente proyecto.

Es por lo anterior que, a la vista de los resultados obtenidos, no cabe duda que la indexación probabilística explora nuevas vías para la investigación en el campo de la clasificación de documentos manuscritos basado en su contenido y que por lo tanto, es una técnica a la que tener muy en cuenta en futuros trabajos.

Además, también hemos explorado otros hiper-parámetros con el objetivo de encontrar diferentes resultados y de estudiar el comportamiento de la red con diferentes valores.

En cuanto a la segunda meta, sobre la exploración del legajo JMDB_4949, podemos afirmar que ha sido cumplida con éxito ya que hemos conseguido clasificar los expedientes que componen tal legajo en sus clases correspondientes. con un bajo porcentaje de error.

Como conclusión, cabe indicar, que gracias a la elaboración de este proyecto, hemos adquirido conocimientos que, por desgracia, en asignaturas como APR (Aprendizaje Automático) no se llegan a ver. Por lo tanto, nos ha servido para mejorar el entendimiento sobre problemas que son resueltos mediante técnicas de *Machine Learning* al igual que el aprendizaje sobre nuevas tecnologías, como lo son los índices probabilísticos.

6.1 Trabajos Futuros

Como mejoras futuras al presente trabajo se presentan las siguientes opciones:

- Segmentar legajo de forma automática, y clasificar: Para el presente proyecto, la segmentación de expedientes ha sido elaborada a mano por especialistas, tarea la cual es larga de elaborar a la par que cara. Debido a esto, una posible mejora al trabajo, seria automatizar la segmentación de los expedientes, y realizar la clasificación tal y como la hemos hecho en el trabajo.

- Exploración de otros legajos: En este trabajo, por cuestión de tiempo, únicamente hemos podido explorar el legajo JMDB_4949, pero, posteriormente, realizaremos la clasificación de expedientes del resto de legajos de la colección.

Bibliografía

[1] Wikipedia - Manuscrito. Consultado en https://es.wikipedia.org/wiki/Manuscrito.

[2] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Natural computation*. vol. 29, no. 9, pp. 2352-2449, 2017

[3] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, "Large-scale image classification: Fast feature extraction and svm training," *in CVPR 2011*. 2011, pp. 1689-1696.

[4] F.Perronnin, J. Sánchez, y T. Mensink, Ïmproving the fisher kernel for large-scale image classification", in European conference on computer vision, *in European conference on computer vision.*. 2010, pp. 143-156. [5], [6]

[5] S. B. Park, J. W. Lee, and S. K. Kim, "Content-based image classification using a neural network", *Pattern Recognition Letter.*. vol. 25, no. 3, pp. 287-300, 2004.

[6] S. Kumar, Z. Khan, and A. Jain, "A review of content based image classification using machine learning approach", *International Journal of Advanced Computer Research.*. vol. 2, no. 3, p. 55, 2012.

[7] Jose Ramón Prieto Fontcuberta, Vicente Bosch y Enrique Vidal. *Textual-content-based classification of untranscribed manuscript images.*. 2020.

[8] Roseblatt, F., Stieber, A., and Shatz, R. H., "The Perceptron: A perceiving and recognizing automaton". *Technical report.* 1958

[9] D.E. Rumelhart, G.E. Hinton, R.J. Williams "Learning representations by backpropagation errors". *Nature.* 323, 533-536, 1986

[10] Hornik, K. .Approximation capabilities of multilayer feedforward networks". *Neural Networks.* 4(2):251–257, 1991.

[11] Vidal, E., Toselli, A. H., and Puigcerver, J. .A Probabilistic Framework for Lexicon-based Keyword Spotting in Handwritten Text Images". *arXiv, Tech. Rep.* 2019.

[12] Bluche, T., Hamel, S., Kermorvant, C., Puigcerver, J., Stutzmann, D., Toselli, A. H., and Vidal, E. "Preparatory KWS Experiments for Large-Scale Indexing of a Vast Medieval Manuscript Collection in de HIMANIS Project". *Proceedings of the International Conference on Document Analysis and Recognition.* ICDAR, 1:311-316, 2017.

[13] G. Salton and C. Buckley "Term-weighting approaches in automatic text retrieval". *" Inf. Proc. Management.* vol. 24, no. 5, p. 513/523, 1988.

[14] T. Joachims .A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization". *" Carnegie-mellon univ pittsburgh pa dept of computer science, Tech. Rep*, 1996.

[15] A. Aizawa "An information-theoretic perspective of tf–idf measures". *"Inf. Proc. Management* vol. 39, no. 1, pp. 45–65, 2003.

[16] T. Bluche, S. Hamel, C. Kermorvant, J. Puigcerver, D. Stutzmann, A. H. Toselli, and E. Vidal, *"Preparatory KWS Experiments for Large-Scale Indexing of a Vast Medieval Manuscript Collection in the HIMANIS Project"* ICDAR, vol. 01, Nov 2017, pp. 311–316, 2017.

[17] A. Toselli, V. Romero, E. Vidal, and J. Sanchez,"Making two vast ´historical manuscript collections searchable and extracting meaningful textual features through large-scale probabilistic indexing". *15th IAPR ICDAR* 2019.

[18] The Carabela Project http://carabela.prhlt.upv.es/es.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," . *Journal of Machine Learning Research* vol. 9, pp. 249–256, 2010.

[20] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *arXiv:1502.03167* 2015.

[21] Nitish Srivastava,Geoffrey Hinton,Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research 15* 2014.

[22] How does Batch Normalization Help Optimization?. Consultado en: http://gradientscience.org/batchnorm/.

[23] Sebastian Ruder, "An overview of gradient descent optimization algorithms" . *arXiv* 2017.

[24] Diederik P. Kingma and Jimmy Ba, "ADAM: A method for stochastic optimization" . *ICLR* 2015.

[25] Duchi, John and Hazan, Elad and Singer, Yoram, "Adaptive subgradient methods for online learning and stochastic optimization" . *Journal of Machine Learning Research* 2011.

[26] Geoffrey Hinton, "Overview of mini-batch gradient descent " . *University of Toronto* 2016.

[27] Adam Paszke and Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *arXiv* 2019.