Extracción de Información en Documentos Antiguos y Manuscritos

Kevin Ortega García

Resumen — El objetivo de este proyecto es el de, partiendo de un set de imágenes de documentos como entrada, realizar una serie de procesos sobre las imágenes con el fin de poder generar un modelo de predicción basado en machine learning que sea capaz de clasificar si los elementos que aparecen en los documentos anteriormente mencionados se tratan de texto escrito a mano, impreso o si no son texto en absoluto. Para ello, se desarrollarán y utilizarán diversos programas, con los que se pretende, por un lado, aislar los elementos de texto de las imágenes y extraer información de dichos elementos, así como crear una matriz de adyacencia que los relacione, y por el otro, aplicar estos datos para entrenar un modelo de predicción que utilice Structured Support Vector Machine. Por último, para comprobar la eficacia de dicho modelo, se harán múltiples pruebas variando los distintos modos de funcionamiento que permite el algoritmo, con tal de observar en qué condiciones funciona mejor, y realizándose un estudio de los mismos.

Palabras clave —Texto manuscrito, texto impreso, text-graphic separation, region properties, centroide, componente conexa, machine learning, support vector machine (SVM), structured support vector machine (SSVM), modelo de predicción, train, test.

Abstract — This project's objective is, starting with a set of document images as input, to carry out a series of procedures on the images with the purpose of obtaining a prediction model based on machine learning able to classify if the elements that appear on the previously mentioned documents are either handwritten text, printed text or no text at all. In order to do that, several programs will be developed and utilized, with which it is intended, on the one hand, to isolate the text elements from the images, extract information of said elements, as well as the creation of an adjacency matrix that relates them, and on the other, to apply this data to train a prediction model that uses Structured Support Vector Machine. Lastly, in order to check the efficiency of said model, multiple tests will be done modifying the various functioning modes that the algorithm allows, with the goal of observing under which conditions does it perform better, and studying the results of those tests.

Index Terms — Handwritten text, printed text, text-graphic separation, region properties, centroid, connected component, machine learning, support vector machine (SVM), structured support vector machine (SSVM), prediction model, train, test.

1 Introducción

Este documento se propone describir el trabajo, desarrollo y conclusiones del proyecto *Extracción de Información en Documentos Antiguos y Manuscritos*, correspondiente al realizado a lo largo de la asignatura *Treball de Fi de Grau* correspondiente a la mención de *Computación* de la carrera *Grau en Enginyeria Informàtica* de la Universitat Autónoma de Barcelona. Originalmente, este tema se trataba de una propuesta realizada por Oriol Ramos Terrades, el que ha sido mi tutor durante la realización del mismo.

A continuación, a lo largo de este documento, explicaré los diferentes aspectos que componen este TFG, comenzando por esta misma sección 1. Introducción donde se sintetizará, en términos generales, en que consiste este proyecto, que se pretende conseguir con él, y cuál ha sido mi trabajo para conseguirlo. Posteriormente, en la sección 2. Objetivos se describirán los diferentes objetivos que se pretenden alcanzar con el proyecto, y el desglose de tareas que se deben realizar para que se cumplan. Después, en el apartado 3. Desarrollo se tratará en profundidad como se ha llevado a cabo el proyecto, explicando detalladamente el trabajo realizado y los conceptos utilizados. En 4. Experimentos se mostrarán los resultados obtenidos de las pruebas realizadas sobre el programa de predicción y como estas han sido ejecutadas, y por último, en 5. Conclusiones se expondrán los resultados del estudio de estos resultados, y en general de las conclusiones extraídas después de la realización de este TFG.

Por lo tanto, este proyecto se propone, a grandes rasgos, el ser capaz de a partir de unas imágenes digitalizadas de documentos, como pueden ser escaneos o fotografías, por un lado, someter estas imágenes a una serie de procesos y algoritmos que permitan aislar las regiones de texto que aparecen en ella, y extraer de las mismas información y datos útiles desde un punto de vista computacional (es decir, datos tales como el tamaño de las regiones o la distancia entre ellas), y por el otro, el utilizar estos mismos datos junto a valores correctos sobre el tipo de texto (manuscrito o impreso) para generar un modelo de predicción basado en el algoritmo *Structured Support Vector Machine* que sea capaz de predecir eficazmente la clase de la región de texto de entre 3, siendo las posibles "texto manuscrito", para bloques de texto escritos a mano, "texto impreso", para bloques de texto creados mediante impresora, imprenta, máquina de escribir u otros medios no-manuales, y por último "sin texto", para bloques que originalmente se habían identificado como regiones de texto pero en las que realmente no lo hay, haciendo esto para cada región de texto identificada en

cualquier imagen o imágenes nuevas que se le proporcionen al programa y de las cuales no se tiene información previa, más allá de la contenida en la propia imagen. Realmente ya se han hecho estudios sobre este concepto [1], pero abordándolo de manera distinta, por lo que resulta igualmente interesante investigar el funcionamiento de un algoritmo como *SSVM* en un problema como este.

Así, mi trabajo ha consistido en, primeramente, estudiar al nivel requerido los algoritmos, programas, y en general, las tecnologías necesarias para llevar a cabo este proyecto como para entender las tareas a realizar y, sobre todo, como hacerlo. Puede sonar trivial, pero realmente esta investigación ha sido una parte imprescindible del proceso de trabajo, ya que, debido a la complejidad de algunas de las tareas, y especialmente, de cómo se tenía que trabajar con diferentes programas y entornos entre sí, se hacía necesario un entendimiento preciso del funcionamiento de los mismos, a la hora de conseguir que estos operasen correctamente y ahorrar la aparición de potenciales problemas futuros.

Por otro lado, el grueso del trabajo se ha concentrado en el desarrollo mismo de los programas necesarios para alcanzar el objetivo del proyecto. Ya que este es conseguir un modelo de predicción funcional, y no la programación de uno desde cero, se han aprovechado herramientas y código ya existentes en la medida de lo posible y que han sido integrados en el proceso. Sin embargo, y como digo, el mayor volumen de trabajo se ha encontrado en esta parte, por lo que igualmente el trabajo de programación a realizar ha sido extenso para lograr un proceso funcional desde la entrada de las imágenes hasta que se obtienen las clasificaciones predichas.

Por lo tanto, las tareas que he tenido que realizar en lo referente a los programas utilizados se pueden dividir en tres partes principales de programación. Por un lado, la utilización de un programa de separación de textográficos, que permiten pasar de la imagen de entrada del documento, a una imagen ya procesada, en blanco y negro y donde solo aparecen las regiones con texto.

La segunda parte consiste en la creación de un programa que identifique espacialmente los bloques de texto de las imágenes generadas anteriormente, así como la extracción de características de los mismos y la generación de la matriz de adyacencia que los relaciona. Esto será explicado con mayor detalle en la sección 3. Desarrollo del documento, pero la información que se obtiene son propiedades tipo el centro de una región de texto, o el área que ocupa. Parte del trabajo a realizar ha sido también el identificar que datos concretos interesaba extraer, y el formato con el que se iban a guardar y con el que se gestionarían.

La última parte de programación se corresponde a la construcción del código que carga los datos generados en el anterior paso, junto a los datos reales de las categorías de los bloques de texto, y que son introducidos en el modelo de predicción para entrenarlo y que aprenda a identificarlos correctamente. También se realiza aquí el proceso de test, que carga información de bloques de texto pero esta vez sin categorías asociadas, e intenta predecir a que clase pertenecen utilizando el modelo entrenado anteriormente, así como la obtención de resultados de estas pruebas.

Por último, la tarea final que he realizado para este *TFG* ha sido la del análisis de los resultados obtenidos fruto de la ejecución de todo el código generado y utilizado descritos en los párrafos anteriores. Para ello se ha realizado una batería de pruebas, variando en cada una de ellas alguno de los parámetros que configuran el funcionamiento del modelo de predicción. Todos estos resultados -estadísticas sobre los aciertos y fallos de las predicciones, así como imágenes mostrando las regiones encontradas y la clase que se les ha asignado-, son extraídos y examinados, con el fin de comprobar bajo qué condiciones el programa funciona mejor.

En lo personal, encuentro que se trata de un proyecto sumamente interesante, ya que utiliza una tecnología que ahora mismo está en pleno apogeo, como es el *Machine Learning*, y me brinda la oportunidad de explorar este campo en mayor profundidad de lo que he podido ver a lo largo de la carrera, al tener que implicarme directamente en un proyecto que lo utilice.

Además, concretamente dentro del extenso concepto que es *Machine Learning*, explora uno de los algoritmos que lo aplican, *Support Vector Machine (*o *SVM* para abreviar*)*, que también había visto ya en algunas de las asignaturas de la mención y con el que he trabajado puntalmente en ellas, pero de manera relativamente superficial, y aplicando sobre ello los *Structured Support Vector Machine (*o *SSVM*, de nuevo para abreviar*)*, una variación más especializada de los SVM, que utiliza las relaciones entre los elementos –*o componentes conexas*- como información adicional para realizar las predicciones, y que aunque personalmente no conocía antes de la realización de este TFG, he podido estudiar y poner en práctica satisfactoriamente.

Ha sido precisamente este aspecto de tener que trabajar y experimentar prácticamente lo que hasta ahora eran solo conceptos teóricos visto en clase, o incluso conceptos nuevos que he tenido que aprender, lo que me ha motivado a lo largo de este proyecto, y lo que me ha permitido llevarlo a buen término.

Por último en esta introducción, quería comentar también que encuentro estimulante el haber trabajado en un proyecto que podría llegar a tener aplicaciones reales. Aunque se trate de una parte pequeña del proceso, y su utilidad tendría que ser comprobada en un entorno real, el trabajo realizado en este TFG se trata de un potencial paso imprescindible en diversos tipos de programa relacionados con identificación y tratado de texto, tales como aplicaciones que transcriban o traduzcan texto a partir de un video. El primer paso a realizar en un programa así sería aplicar un proceso similar al que se ha trabajado en este TFG, ya que la forma de tratar el texto y los algoritmos que se le tendrán que aplicar para la transformación serán distintos en función de si este es manuscrito o impreso.

2 OBJETIVOS Y TAREAS

Tal y como se ha descrito anteriormente, el objetivo principal de este TFG es el de hacer un programa, que mediante un modelo de predicción basado en *SSVM*, y dadas unas imágenes de documentos como entrada, sea capaz de identificar y clasificar si el texto que aparece en estas imágenes se trata de texto manuscrito o impreso. Si bien este es efectivamente el objetivo principal del proyecto, debido a la complejidad del mismo, se ha hecho necesaria desglosarlo en varios sub-objetivos que deberán ser satisfechos antes de poder considerarlo completado.

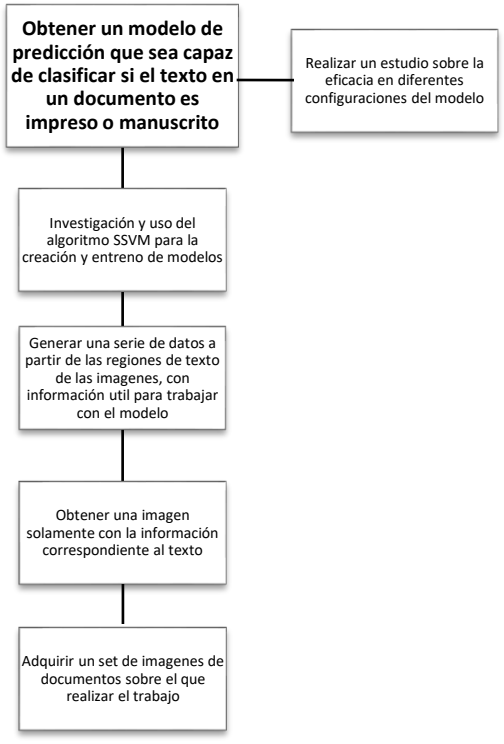


Fig. 1. Diagrama con los objetivos a alcanzar en el proyecto. En la parte superior aparece el objetivo principal, y en orden descendiente los que se deben satisfacer antes de este

Así, en primer lugar se debe conseguir un set de imágenes sobre las que se trabajará a lo largo del proceso. Junto a estas imágenes se tiene que poseer datos reales sobre los bloques de texto y las clases a la que pertenecen, con el propósito de poder entrenar posteriormente el modelo con información que se sabe que es correcta. De estas imágenes, se tienen que obtener una nueva imagen por cada una de ellas conteniendo únicamente las secciones de texto, eliminando en el proceso la parte grafica que no tiene utilidad para el proyecto. Ya con la parte de las imágenes que podemos aprovechar, se tiene que extraer de ellas una serie de datos relativos a las regiones de texto, con las que luego se entrenará el modelo, y que también servirán para hacer las predicciones de las mismas.

Dada la dificultad en el uso del algoritmo SSVM, se tiene que hacer un estudio previo a su uso, sobre cómo funciona, que tipo de datos usa y de qué manera, así como que parámetros permiten configurarlo, y también de cómo trabajan las funciones de entrenamiento y predicción. Una vez todos estos conceptos están claros, ya se puede pasar al objetivo principal, que como se explicaba antes, es el de conseguir un modelo de predicción funcional. Como resultado de haber completado el objetivo principal, se pueden extraer los resultados del funcionamiento del programa, con el propósito de analizar el rendimiento y funcionamiento del mismo, y así poder extraer unas conclusiones con las que finalizar el TFG.

No obstante, si bien estos objetivos funcionan en un plano teórico, es necesario desempeñar un trabajo, ya sea de investigación o de implementación en código que permita alcanzarlos. De nuevo, debido al volumen de este trabajo, se ha desglosado este proceso en una serie de tareas y sub-tareas que se han ido realizando a lo largo de la duración del TFG, y que serán descritas a continuación. No obstante, para una mejor comprensión de las mismas, y para poder visualizar mejor el proceso, se recomienda consultar en la sección de apéndices, el apéndice A1. Distribución de tareas, donde se muestra una tabla con la disposición concreta de las tareas (A1.1), y un diagrama de Gantt (A1.2) donde se puede ver la distribución temporal de las mismas.

Las tareas y sub-tareas se dividen por lo tanto de una manera similar a como se ha hecho con los objetivos, ya que como norma general cada una de las tareas principales pretende satisfacer al menos uno de los sub-objetivos. El cómo se han llevado a cabo estas se explicará en detalle en la siguiente sección 3. Desarrollo, pero en términos generales se pueden definir en los siguientes bloques: hacer funcionar el programa que separa la imagen del documento original en una imagen con el texto, y otra con los elementos gráficos.

Extraer las características de la imagen de texto, identificando las áreas de texto y las componentes conexas en el proceso. Originalmente esta tarea se había planificado hacerse codificando el programa en Matlab. No obstante, finalmente se decidió traspasar el progreso y terminarlo en Python. Posteriormente se adaptaron estos programas para que en lugar de simplemente servir para probar su funcionamiento con una imagen, ya funcionasen de manera definitiva y correctamente con tantas imágenes como fueran necesarias.

Por último, se tiene que implementar el programa que utiliza los *SSVM*, así como la gestión de los datos que se le deben aplicar, y la extracción de los resultados, de los que se realizara un estudio una vez se hayan hecho las pruebas pertinentes. Por otro lado, y paralelamente a las tareas enumeradas anteriormente, se irá redactando la documentación necesaria para el TFG a tiempo para las entregas pertinentes.

3. DESARROLLO

En esta sección se describirá detalladamente todo el trabajo realizado a lo largo de este TFG, así como especificaciones sobre los programas utilizados, y descripciones

más exhaustivas sobre los algoritmos y conceptos utilizados para llevar a cabo los objetivos del proyecto. De este modo, se explicarán las tareas que se han realizado, siendo estas agrupadas en las subsiguientes sub-secciones divididas por bloques temáticos que satisfacen al menos uno de los objetivos o sub-objetivos expuestos anteriormente.

3.1 Eliminación de ruido y elementos gráficos de la imagen

En esta parte el propósito era conseguir procesar una imagen de entrada, que, para el objetivo de este TFG, es una copia digital (escaneo o fotografía) de un documento tal y como se ve con su aspecto real y físico, y extraer de ella solamente las regiones de la imagen con texto, en blanco y negro. Para ello, se utiliza un programa escrito en C++ que, aplicando un algoritmo de separación de texto-gráficos, permite aislar la parte que nos interesa, la de texto [2]. De hecho, como resultado de la ejecución del programa, se obtiene una imagen tanto de la parte gráfica como de la de texto. No obstante, la primera se descarta ya que no tiene utilidad para el trabajo que se pretende realizar en este proyecto; sin embargo, la imagen con texto será la base sobre la que se trabajará a lo largo de este TFG.

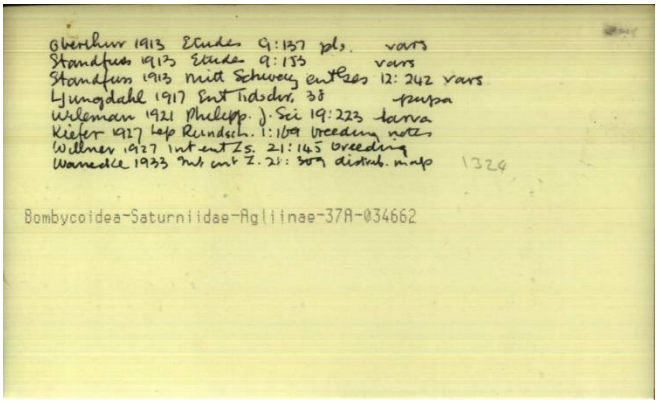


Fig. 2. Imagen original de un documento, antes de aplicarle ningún proceso.

Aun así, los resultados pueden no ser del todo perfectos, y puede que lleguen elementos gráficos o de ruido a la imagen de texto. Esto no es un problema, ya que entre los objetivos del TFG no se encuentra el de perfeccionar un algoritmo de separación de texto-gráficos, por lo que la pureza que nos da este programa ya cumple su propósito, que es el de proporcionarnos con una imagen de texto sobre la que trabajar y demostrar que se puede aplicar un algoritmo de *SSVM* en un caso como este.

Cabe además decir que este código no fue realizado por mí, sino que me fue dado al comenzar el proyecto del TFG, de nuevo por el mismo motivo de que mencionaba anteriormente; no merece la pena tener que invertir tiempo y esfuerzo en aprender a programar un algoritmo que permita realizar esta tarea, cuando al fin y al cabo no forma parte del propósito de investigación de este proyecto.

Por lo tanto, el trabajo realizado en esta parte no ha sido el de la construcción del programa, sino el de, por un lado, conseguir que compilase, tarea que ha resultado más complicada de lo esperable debido a problemas de compatibilidades entre librerías, y el de adaptar el programa para emplearlo en este TFG. Concretamente, el problema de librerías ha sido causado por la librería de OpenCV para el entorno IDE utilizado, Visual Studio, y C++, el lenguaje utilizado en el programa.

Originalmente se dieron abundantes problemas a la hora de cargar las librerías adecuadamente, lo que causó que el programa no funcionase en absoluto. Afortunadamente, después de múltiples intentos con diferentes configuraciones y versiones tanto del IDE como de las librerías, se consiguió que compilase de manera correcta, tras lo cual solo quedó el adaptar el programa para que cargase las imágenes deseadas, y así obtener la imagen solamente con las regiones de texto, lo cual era el propósito del mismo, y con lo cual finalizaba este bloque.

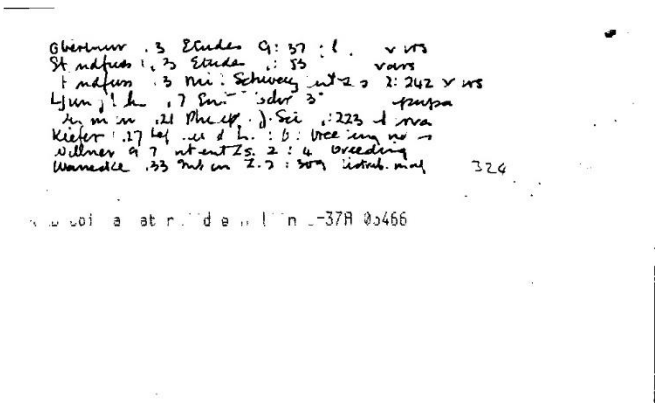


Fig. 3. Imagen resultante de texto después de aplicar el programa de separación texto-gráficos a la imagen original.

Por si fuera de utilidad sobre el conflicto anterior, o como mínimo para aportar información adicional sobre el cómo se ha realizado esta parte, la versión de Visual Studio que ha sido utilizada es, concretamente, Visual Studio 2012 32 bits. Por otro lado, la versión de OpenCV es la 2.4.13. El sistema operativo utilizado ha sido Windows 10 Education 64 bits.

A pesar de que no conozco el funcionamiento de este programa tan bien como si lo hubiera hecho desde cero, debido a esta problemática que surgió y al trabajo que se tuvo que realizar en él, fue necesario examinar el funcionamiento general del código, y por lo tanto alcanzar un grado de comprensión y familiarización razonables sobre que procedimientos se seguían, y como se debía utilizar.

3.2 Adecuación de datos y programas

En este bloque se agrupan una serie de tareas que se hicieron necesarias al alcanzar cierto punto de progreso en el proyecto. Por lo tanto, el trabajo realizado aquí afecta a algunos de los otros bloques del desarrollo, pero se co-

mentan aquí debido a sus características comunes.

Básicamente, se tuvieron que realizar varias modificaciones en el código que ya había hecho, tanto de la parte 3.1 como en la 3.3, que será descrito en la siguiente sección. Debido a que en un principio se buscaba la funcionalidad general de los programas, pero no necesariamente que tuvieran el funcionamiento definitivo, originalmente se habían hecho funcionar para una sola imagen. No obstante, el propósito final de este TFG es el de trabajar con *n* imágenes, donde *n* es como mínimo dos, pero que puede ser una cantidad tan grande como se desee. Por ello, se tuvo que adaptar el código para que pudiese funcionar con tantas imágenes como con las que fuera requerido trabajar. Además, se realizaron pequeños arreglos y ajustes adicionales que se descubrieron al revisarlo.

Por otro lado, también se hizo aparente la necesidad de utilizar información de *groundtruth*, y que no se había generado hasta el momento. Para ello, se usó una serie de archivos *xml*, habiendo uno por cada una de las imágenes que se planeaban utilizar como dataset a lo largo del proyecto. En dichos archivos había información concreta y precisa sobre las áreas de texto de la imagen, y sobre si estas pertenecían a texto manuscrito o texto impreso. Por lo tanto, a partir de las dimensiones de la imagen original y este archivo se genera una imagen alternativa de fondo negro donde no hay texto, y donde aparecen resaltadas en color azul las regiones con texto manuscrito y en verde las regiones con texto impreso.

Estas imágenes se utilizarán por un lado como referencia para el entrenamiento del modelo, ya que para que aprenda a categorizar necesita conocer a que clase pertenecen los elementos con los que se está entrenando, y por el otro para cuando posteriormente se quieran analizar los resultados de las predicciones, ya que se tienen que contrastar con los resultados que se sabe que son correctos.

Fig. 4. Imagen de *groundtruth* generada a partir del archivo *xml* correspondiente. Nótese las áreas azules, que corresponden a texto manuscrito, y la verde, correspondiente a texto impreso.

3.3 Extracción de características de la imagen

Este bloque se trata de una parte crítica del proceso, ya que aquí es donde se extrae toda la información de las imágenes con la que luego tiene que trabajar el modelo, tanto para el entrenamiento como para las predicciones.

Originalmente se planteó hacer esta parte creando un programa en Matlab. Efectivamente, así se hizo, parcialmente al menos, pero debido a que por un lado, a pesar de que lo conocía y ya había trabajado con él con anterioridad, no estaba tan familiarizado con Matlab como con otros lenguajes, y que de todas formas los datos generados en este bloque iban a ir a parar a un programa hecho en Python (descrito en *3.4 Modelo de predicción*), por lo que era mejor trabajar con el mismo tipo de datos, se decidió que finalmente sería mejor realizar esta parte directamente en Python, trasladando el código ya hecho en Matlab a este lenguaje, y añadiendo a posteriori las funcionalidades que faltaban.

Así, la información que se extrae de las imágenes, y la manera de la que se obtiene, es la siguiente:

Propiedades de las componentes conexas, obtenidas a partir de un algoritmo que busca automáticamente las regiones de interés de una imagen, y que proporciona información y características de las mismas. Para el propósito de este TFG, las que se han considerado de interés son:

1) *Centroide*; coordenada (x, y) del punto donde se encuentra el centro de la región.
2) *Área*; área total de la región en pixeles.
3) *Bounding box*; cuadro delimitador de la región. Viene definido de la siguiente manera: mínima posición horizontal, mínima posición vertical, máxima posición horizontal, máxima posición vertical.

Cabe decir que todos los valores de la componente conexa han sido normalizados para ajustarse a un espacio 0,1 (es decir, las coordenadas x, y mínimas serían (0,0), mientras que las máximas serían (1,1)), para así ser validas siempre y no depender del tamaño de la imagen original. Por otro lado, a la información de las componentes conexas se le añade este último dato:

4) *Color*; 0 si en el pixel del centroide no hay texto (aparece blanco) o 1 si sí lo hay (aparece negro)

Con esto, se prepara toda la información que representa la componente conexa, y sobre la que después se realizarán las predicciones.

Precisamente por eso, paralelamente a la identificación de estas, se busca y se guarda una lista a que clase pertenecen realmente en la imagen groundtruth correspondiente, buscando la coloración en dicha imagen en el punto donde está ubicado el centroide de la componente conexa, siendo la codificación de nuevo azul para texto manuscrito, verde para impreso, y negro para no-texto.

Una vez terminados los pasos anteriores, tenemos una serie de datos guardados en una lista, pero que realmente todavía no son las componentes conexas que buscamos, al ser regiones independientes entre ellas y que no se relacionan de ninguna manera. Es por ello, que en el siguiente paso aplicamos a la lista de centros un algoritmo que crea una red con los puntos formando una triangulación de Delaunay. Esto nos asegura que cualquier centroide esté al menos relacionado con otros dos.

Esta triangulación viene dada estructurada como una lista formada por los diferentes triángulos que forman la

triangulación. Como para el propósito de este TFG se requieren las conexiones individuales entre dos puntos, se almacenan cada una de las relaciones de manera recíproca (es decir, tanto la relación del centroide A con el B, como la de B con A).

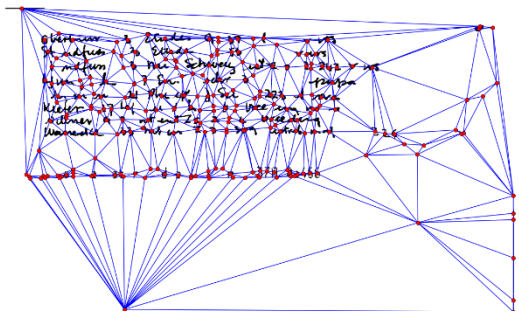


Fig. 5. Imagen de texto, con los centroides de las regiones encontradas superpuestos, y con las aristas de la triangulación de Delaunay.

Por último, para poder añadir algo de información sobre las aristas de las relaciones, se calcula la longitud de las mismas, de nuevo, normalizadas, y se almacenan para luego utilizarlas también en el aprendizaje y entrenamiento del modelo.

Como nota, añadir que las características y propiedades extraídas en este bloque no son algo fijo; se ha decidido trabajar con estos datos para el propósito de este TFG, no obstante, es posible añadir nuevas propiedades que pudieran ser relevantes o eliminar alguna que pudiera considerarse redundante o innecesaria. Esto no impediría el funcionamiento del modelo con SSVM siempre y cuando se mantuviera un formato adecuado, aunque sí que afectaría a los resultados. Para comprobar si estos potenciales cambios tienen efectos positivos o negativos sobre la precisión de las predicciones, se tendrían que realizar pruebas adicionales.

3.4 Generación del modelo de predicción

Por último, aquí se describe el desarrollo del que es el propósito principal de este TFG, que es el de, por un lado, preparar un modelo de predicción que aplique el algoritmo *SSVM*, por el otro, entrenar dicho modelo con parte de la información generada como resultado del trabajo realizado en el bloque 3.3 y las etiquetas obtenidas a partir del *groundtruth*, y para finalizar, el de efectuar la predicción propiamente dicha, sobre la otra parte de los datos, pero esta vez sin aportar la información de las clases reales, ya que es lo que debe predecir el modelo. Los resultados de dichas predicciones son entonces evaluados, para conocer con qué grado de precisión a clasificado la información proporcionada. Este último paso será examinado más exhaustivamente en la sección 4. Experimentos.

Debido a que esta tarea entraña una complejidad y dificultad elevadas debido al funcionamiento intrínseco del algoritmo, y al hecho de que esto se complica todavía más por el hecho de que a pesar de haber trabajado anteriormente de una manera superficial con el algoritmo *Support Vector Machine*, no había trabajado nunca con esta derivación del mismo y, de hecho, ni siquiera la conocía con anterioridad al inicio de este proyecto, se ha optado por utilizar como referencia un ejemplo que lo aplicaba, *Semantic Image Segmentation on Pascal VOC* [3]. Este ejemplo es esencialmente distinto de lo que se pretende conseguir con este TFG, tanto en la estructura de los datos que se utiliza como en los datos en sí mismos, así como en las predicciones que se quieren conseguir, pero el funcionamiento en si del modelo SSVM es fundamentalmente el mismo, y por ello ha servido para entender mejor como funciona, que parámetros de configuración específicos requiere, y para qué sirven.

La diferencia más significativa entre un algoritmo SVM y el SSVM, siendo este último el utilizado en este TFG, es que el SSVM admite estructuras de datos más amplios, más abstractos. Para el caso que nos ocupa, la principal divergencia respecto a SVM es que aquí le proporcionamos al modelo no solo los elementos a clasificar, sino una matriz de adyacencia que los relaciona.

Como información adicional, tanto este bloque como el trabajo realizado en el bloque 3.3 se ha optado por que se hiciera utilizando el sistema operativo Ubuntu 16.04.1 LTS 64 bits, a diferencia del resto, ya que originalmente se había planteado trabajar a lo largo del proyecto en Windows. Este cambio es debido a la necesidad de añadir la librería pystruct [4], que nos da las herramientas necesarias tanto para la creación del modelo como para su configuración, y que por lo tanto resulta imprescindible para lograr alcanzar el objetivo de este proyecto. A su vez esta librería tenía una serie de dependencias con otros complementos de Python que debían ser instalados previamente. Uno de estos paquetes de software era CVXOPT [5], cuya instalación resultó ser extremadamente complicada en sistemas operativos Windows, hasta el punto de realizar una inversión de tiempo considerable sin éxito alguno. Por el contrario, se comprobó que en Ubuntu se podían instalar todas las librerías requeridas y que el ejemplo que se ha usado como referencia funcionaba correctamente. Por ello se optó por trabajar en dos sistemas operativos distintos, a pesar del inconveniente que esto supone, antes que invertir un tiempo muy valioso en resolver problemas de compatibilidad.

Entrando en mayor profundidad en el funcionamiento del programa, y debido al alcance del trabajo realizado en este bloque, hay varias partes con un funcionamiento diferenciado. Para empezar, es necesario el cargar todos los datos generados en el bloque 3.3 de una manera específica. Cabe además destacar que estos datos serán cargados y organizados en listas como se explicará a continuación, y que estas listas no son los *arrays* habituales de Python, sino que el entrenamiento del modelo de predicción fuerza que los datos estén guardados en *numpy arrays*, un formato especial de lista característico de la librería *numpy*, por lo que cada vez que se hable de lista se ha de suponer o bien que se trata de un *numpy array*, o bien que eventualmente será transformado a este tipo.

Por un lado, se necesitan agrupar todas las características de las componentes conexas en una lista. El orden es indistinto, pero por cada componente conexa sus caracte-

rísticas deben estar dispuestas de manera secuencial. Así, cada elemento de la lista se corresponde a una componente conexa completa, y habrá tantas filas como componentes conexas haya. Por otro lado, se deben almacenar las relaciones de la triangulación de Delaunay. Estas se guardan en una lista, donde cada fila es la pareja de componentes conexas, que vienen indicadas por su posición en la lista anterior; es decir, que por ejemplo una relación 34, 57 significaría que las componentes conexas que se encuentran en las posiciones 34 y 57 tienen una conexión entre ellas. De nuevo, la lista tendrá tantas filas como conexiones haya. El último elemento de esta lista son las propiedades de las aristas de las conexiones. En nuestro caso es solamente una, la longitud de la arista, y por supuesto, se ha de tener una longitud para cada una de las conexiones, por lo que la cantidad de elementos en ambas listas debe coincidir.

Por el otro, se han de cargar las clases (texto manuscrito, impreso, no-texto) correctas de las componentes conexas que se extraían de las imágenes groundtruth, también el bloque 3.3. Cada componente conexa ha de tener su clase correspondiente, por lo que ambas listas han de tener el mismo tamaño.

Esto se ha de realizar cada vez por cada una de las imágenes que compongan la muestra de datos, en nuestro caso, las imágenes de los documentos, y por lo tanto se terminará con dos listas, una con las componentes conexas, conexiones y propiedades de las aristas, y otra con las clases, ambas de n elementos, donde n es la cantidad de imágenes totales que se quieren utilizar, y cada uno de los elementos se corresponde a los datos de una imagen en concreto.

No obstante, para el propósito de este programa, es necesario separar estos datos en segmentos de train y test; esto es, que una parte de los datos se utilizará para entrenar el modelo de predicción, y la otra se utilizará para que haga las predicciones sobre ella y así poder comprobar su funcionamiento. Por ello, se ha aplicado *k-fold cross validation*; es decir, se ha dividido el dataset en k partes, utilizando en cada iteración una de ellas como datos de test, y el resto como datos de entrenamiento, alterando en cada iteración la parte utilizada para test y train de manera que no coincidan entre iteraciones. Para este TFG se ha utilizado $k=5$, por lo que un 20% de la información ha sido destinada a test y un 80% a entrenamiento para cada prueba, pero este valor se puede cambiar fácilmente al que se requiera.

Una vez con los datos preparados, pasamos a la configuración del modelo. Debido a la naturaleza limitada del TFG, la mayor parte de parámetros que permiten modificación en el modelo, se han dejado, o bien, con los valores por defecto, o con los sugeridos en el ejemplo de *SSVM* que se ha utilizado, o bien no se ha introducido ningún parámetro si el propio modelo es capaz de deducirlo en función de los datos proporcionados.

Sin embargo, hay algunos datos, muy relevantes para el funcionamiento del modelo, con los que sí que se ha trabajado y se han configurado manualmente. Estos son:

1) El *número de clases*, que para este TFG son siempre 3, texto manuscrito, texto impreso y no-texto.
2) El *peso* de las clases, que se calcula en base a una función matemática que considera el número de apariciones de cada clase en la muestra, y que determina que predisposición hay en el aprendizaje a asignar unas clases u otras.
3) *C*, o *parámetro de regularización*, que determina el ratio de aprendizaje del modelo.
4) El *método de inferencia*. Este parámetro admite cuatro tipos de método de inferencia; *max-product, lp, ad3, qpbo*. Para los resultados finales, se han hecho pruebas con todos ellos, no obstante, con *lp* estas han sido solo parciales, ya que el tiempo de ejecución con este método es desproporcionadamente superior al de los otros, y se ha considerado demasiado costoso en términos de tiempo.

Una vez está el modelo correctamente configurado, ya se puede proceder al entrenamiento del mismo. Para ello, se le introducen los datos de train que previamente han sido preparados. Así, el programa comienza el proceso de aprendizaje. Este proceso es automático, y a lo largo de él, se van realizando iteraciones hasta que se consigue alcanzar convergencia en sus resultados internos. Cuando pasa esto, el entrenamiento ha finalizado, y se puede proceder al siguiente paso.

Para ello, se introducen los datos de test, con el propósito de que el modelo sea capaz de predecir a que clase pertenecen las componentes conexas que se le proporcionan. Una vez el programa termina de hacer las predicciones, obtenemos una lista de valores correspondientes a la clase asignada a cada una de las componentes conexas, finalizando así el programa, y con ello concluyendo y cumpliendo el objetivo principal de este TFG.

Sin embargo, todavía resta una parte más del proyecto, que es la de analizar estos resultados, comparando las clases predichas con las reales, y extrayendo varias estadísticas relativas a su eficacia y su precisión. Por lo tanto, se han realizado diversas pruebas con configuraciones distintas, en pos de conocer mejor el funcionamiento del algoritmo, y como afecta la configuración a los resultados. Este proceso será descrito en la siguiente sección del documento, 4 Experimentación.

4 Experimentación

En esta sección se describirán las pruebas y experimentos realizados, una vez se ha conseguido el correcto funcionamiento del programa. Así, se ha llevado a cabo una serie de pruebas sobre las cuales se mostrarán y comentarán los resultados aquí, y se realizará un estudio de las misas cuyas conclusiones serán expuestas en la sección 5 Conclusión.

Respecto a las pruebas, se ha preparado una serie de tests consistentes en variar la configuración del modelo. Así, para cada uno de los modelos de inferencia, *max-product, lp, ad3* y *qpbo* se ejecutará el programa de predicción completo, probando además para cada uno de ellos 4 valores de *C*, o parámetro de regularización, siendo estos 0.01, 0.1, 1 y 10, realizando así en teoría un total de 16 pruebas.

Sin embargo, finalmente este número ha sido menor, ya que como se ha comentado anteriormente, el método de inferencia lp es excesivamente costoso tanto a nivel de recursos en el equipo utilizado, como en lo referente al tiempo de ejecución, por lo que solo se ha hecho una prueba con este modelo usando C=0.1, reduciéndose así el total de pruebas a 13. Además, cabe recalcar que esta prueba con lp ni siquiera es completa, habiéndose podido terminar solamente la primera iteración de la técnica *k-fold*, por lo que se han utilizado las primeras 20 imágenes como test, y las restantes 80 como train, pero no existiendo una validación cruzada como tal.

Por lo tanto, el proceso seguido para realizar los experimentos es el siguiente: el programa que construye y entrena el modelo está preparado para que, previa designación manualmente del modelo de inferencia que se va a utilizar, este proceso se ejecute automáticamente tantas veces como diferentes valores de *C* se hayan escogido, y para cada uno de estos valores, tantas veces como el valor que tenga la *k* determinada para el *k-fold*, con el fin de probar tantas variaciones del modelo con distintos entrenamientos como divisiones se hagan del conjunto de datos. Así, para el caso que se ha utilizado para las pruebas, se utilizarían 5 sets de datos de test distintos entre ellos, y diferentes a los de train para cada vez, funcionando a la práctica, efectivamente, como 5 conjuntos de datos distintos, a pesar de trabajar con el mismo set de imágenes en todo momento.

Respecto a la estructura de datos, a diferencia del train, donde se le proporcionaban al algoritmo las parejas de datos constituidas por las componentes conexas (y sus relaciones y propiedades asociadas) y sus etiquetas (clases) correspondientes, para train se utiliza solamente la parte relativa a las componentes conexas (y de nuevo, sus relaciones y propiedades asociadas), ya que el objetivo es precisamente que estas etiquetas se obtengan a partir de las predicciones. Sin embargo, las clases reales de estas componentes conexas sí que se utilizan posteriormente para compararlas con las clases predichas, y comprobar así la precisión en el funcionamiento del modelo.

Esto se hace aplicando una función, que a partir de las listas con las etiquetas correctas y las predichas, devuelve una serie de estadísticas extraídas para cada una de las clases por separado, que son:

1) *Precisión*, el porcentaje calculado en base al total clases predichas correctamente sobre el total de representantes real de una clase.
2) *Recall*, o el porcentaje de elementos de predichos para una clase que realmente se corresponden a la clase que se les ha asignado.
3) *Fbeta score*, la media armónica ponderada entre la precisión y el recall

Estas estadísticas se generan para los resultados de cada una de las imágenes individualmente, pero también para los resultados de cada una de las iteraciones del *k-fold*, y por último, una vez más para las predicciones globales después de utilizar el dataset completamente al usar todos los sub-conjuntos de datos como test. Estos datos son útiles, ya que permiten saber con exactitud el rendimiento del modelo en cada momento de la prueba, y por ello mismo son sobre los que se realizará el estudio del que posteriormente se extraerán las conclusiones. Toda esta información se almacena de manera organizada en archivos con formato *csv*, uno para cada prueba, de manera que se facilite su consulta, así como la gestión de los datos, como para por ejemplo comparar resultados entre varias pruebas, o generar gráficas de las estadísticas (se puede ver una muestra de estas graficas en el apéndice A2. Resultados de Experimentos).

Paralelamente a esto, y para poder contar con toda la información posible, también se generan unas imágenes (Fig. 4) similares a las que se crearon anteriormente para ser usadas como *groundtruth*, pero en este caso utilizando las bounding boxes de las componentes conexas como regiones a destacar en la imagen, y las clases determinadas por el modelo para determinar el color, existiendo una imagen con las predicciones por cada una de las imágenes que se habían introducido originalmente. De este modo, aparecerán en rojo las zonas predichas como no-texto, en azul las de texto manuscrito, y en verde las de texto impreso. De esta manera resulta mucho más fácil comprobar los resultados de manera visual, y se hace más aparente el grado de eficacia de las predicciones, al menos de manera general, al poder compararlo con las imágenes de *groundtruth*.

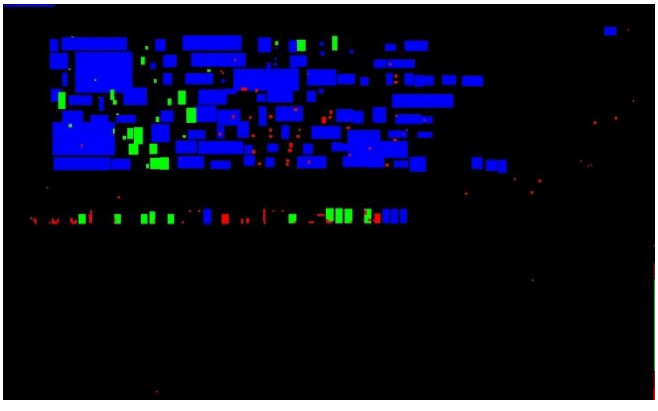


Fig. 6. Imagen con las categorías pertenecientes a las predichas por el modelo, para usando ad3 como método de inferencia y C=0.1. Nótese la composición similar a la imagen de groundtruth para la misma imagen original (Fig. 4)

Por lo tanto, una vez con todas las pruebas terminadas, y con la información recopilada y debidamente examinada, se puede proceder a su análisis, y con ello, a las conclusiones fruto del trabajo realizado en este proyecto, que serán explicadas en la siguiente sección del documento, 5 Conclusión.

5 Conclusión

Una vez llegados a este punto, lo primero que cabe destacar es que, tal y como se había propuesto comprobar como objetivo en este TFG, efectivamente se ha demostrado que se puede, aplicando un modelo de predicción basado en Structured Support Vector Machines, resolver el problema de identificación de regiones de texto, y si estas son de texto manuscrito o no, en documentos antiguos y/o

manuscritos. No obstante, y en base a los resultados de las pruebas obtenidos, ahora resta comprobar en qué grado funciona adecuadamente este sistema, y hasta qué punto es válido como solución.

Y es que los resultados no son tan buenos como quizás cabría esperar. Generalizando, y es que los resultados pueden variar considerablemente de prueba a prueba, o incluso dentro de la misma, la *precisión* de los resultados suele rondar el ~45% en el mejor de los casos para las regiones manuscritas, el 50% para el texto impreso y el 30% o menos para las zonas que no tienen texto. Mientras tanto, respecto al *recall*, este suele rondar el 60% para la clase no-texto, 50% para las regiones de texto manuscrito y el 30% o menos para las de texto impreso. Esto significa que se está dejando como mínimo la mitad de elementos por predecir correctamente para cualquier clase, y que las predicciones que hace alrededor de una de cada dos veces no pertenecerá correctamente a la clase que le corresponde, sino a una de las otras dos. Estamos por lo tanto en algunos casos ante unos porcentajes que no se alejan mucho de la distribución normal para tres clases, si las predicciones fueran hechas aleatoriamente (un ~33%), lo cual es preocupante, ya que esto querría decir que aparentemente el modelo no está aprendiendo correctamente a realizar las predicciones.

Si examinamos más detalladamente los resultados por pruebas, vemos diferencias significativas entre los diferentes modelos de inferencia. Primeramente, *ad3* y *qpbo* dan unos resultados razonablemente parecidos, y que se ajustan bastante a las estadísticas mencionadas anteriormente. *Ad3* parece ser más consistente entre resultados de diferentes pruebas y clases, mientras que *qpbo* da valores más extremos, tanto positivos, dando algunos de los mejores resultados para la precisión en la clase 1 como el recall en la clase 0, como negativos, dando también algunos de los peores resultados.

Por otro lado, el método *max-product* no parece funcionar adecuadamente. Usando cualquier configuración con max-product implica que todas las componentes conexas son etiquetadas con la clase 0, o no-texto. Por lo tanto, los resultados obtenidos aplicándolo no son válidos en ningún caso, y no es viable su uso.

Por último, en lo referente a métodos de inferencia, *lp* da unos resultados sorprendentes. Por supuesto, las pruebas con este método han sido mucho menos exhaustivas que con los demás, y al haber utilizado un solo subconjunto de datos, podrían obtenerse de él unos resultados muy específicos para el set de train con el que ha hecho el aprendizaje. No obstante, los resultados son en general bastante comparables a los obtenidos con *ad3* y *qpbo*, pero obteniendo en algunos casos unos picos de *precisión* y *recall* que exceden los obtenidos en ninguna otra prueba. Aun así, con los recursos físicos que dispongo, y los de tiempo que me impone el proceso del TFG, me ha resultado imposible hacer más pruebas para este método.

Por otro lado, es reseñable el destacar también que variar el valor del *parámetro de regularización* no parece tener efectos demasiado significativos en los resultados. Si que se aprecian algunos pequeños cambios en algunos casos en las imágenes individuales, pero estas variaciones no llegar a afectar de manera notable la media de los resultados, que suele rondar un ±1% entre todos los posibles valores de *C* para *ad3*, y afectando algo más al método *qpbo*, sin, de nuevo, cambios demasiado relevantes, a excepción de *C=0.01*, que si altera de manera más apreciable los resultados.

Examinando los resultados individuales que se generan a lo largo de cada una de las pruebas, se pueden encontrar otros comportamientos inusuales. El que más llama la atención, y que se hace evidente nada más ver las imágenes con las clases predichas, es que hay fuertes cambios en las predicciones cuando se hace una iteración nueva del *k-fold* y por lo tanto cambia de conjunto de train, para incluso dentro de una misma configuración.

Concretamente, parece que para cada subset de entrenamiento se genera un fuerte sesgo en las predicciones, haciendo que se decante desproporcionadamente más por unas clases que por otras. Por ejemplo, en la primera iteración, suele mostrar una predisposición muy notable a categorizar las componentes conexas como texto manuscrito, mientras que para la segunda la mayoría de predicciones las hará en favor de texto impreso.

Esto pareciera indicar que existe *underfitting* en el modelo tras el aprendizaje, ya que, por un lado, hay que considerar que el set de datos con el que he trabajado es muy pequeño y que por el otro explicaría estos resultados que claramente adquieren una tendencia hacia una clase u otra a pesar de que la diferencia en los conjuntos usados para entrenar es pequeña, y es que parece que el modelo no está aprendiendo a clasificar adecuadamente, sino que simplemente establece unas restricciones estrictas que favorecen a una clase en específico y que rara vez se cumplen para las otras. Esto sucede tanto para el método *ad3* como para el *qpbo*, y en *max-product* no se aprecia al estar todo categorizado como no-texto.

Sin embargo, resulta interesante observar que, a pesar de que no se han realizado pruebas suficientes como para ver si esto sucede también con el método *lp*, no parece que esto suceda también con él, o por lo menos no con la misma severidad. A pesar de que solo se tienen resultados del primer subset, que como he comentado antes parece producir una tendencia hacia la clase texto escrito, *lp* da la precisión más alta de entre todos los modelos para una de las clases discriminadas por *ad3* y *qpbo* en este caso, texto manuscrito, aunque también es cierto que obtiene un recall especialmente bajo para esta clase.

Una vez estudiados los datos, y examinado detenidamente el comportamiento del modelo con *SSVM* frente al problema de la categorización, estas son las conclusiones que he podido extraer. No obstante, teniendo claros cuales son los puntos positivos del proyecto y cuales sus fallos, también se hace aparente una línea de trabajo futura a seguir, ya que si bien los resultados distan de ser excelentes, y por supuesto el programa no se puede utilizar tal y como está ahora para entornos reales, sí que es

cierto que el modelo como tal funciona, y en algunas pruebas, en imágenes concretas, consigue dar buenos resultados, por lo que considero que puede merecer la pena seguir trabajando en intentar resolver este problema mediante SSVM para alcanzar un mayor grado de satisfacción y eficacia.

Para ello, a partir del análisis de los resultados y de las mismas conclusiones, he podido inferir una serie de protocolos, cambios o simplemente investigaciones que se podrían aplicar al programa con el fin de buscar esta mejoría en su funcionamiento. Debido a las limitaciones propias del formato del TFG no ha sido posible estudiar a la práctica estas sugerencias, pero en caso de querer retomar el progreso de este proyecto, estas son varias potenciales líneas de mejora a seguir.

Partiendo de la base, sería interesante cambiar de algún modo el dataset utilizado a lo largo de todo el proceso, ya sea aumentando el tamaño del mismo, y es que considero que probablemente uno de los principales problemas sea que tenemos muy poca información sobre la que trabajar, variando los tipos de imágenes que hay en él, haciendo que, por ejemplo, sean más parecidas o diferentes entre sí y ver cómo se comporta el modelo frente a estos cambios, o mejorando la calidad de las imágenes. Esto último lo menciono no porque la calidad de las imágenes sea mala per se, pero sí que he notado que, en algunas de las imágenes, en los extremos aparecen elementos que no corresponden al documento, pareciera ser por estar mal cuadrados en la fotografía, y pudiéndose ver lo que hay más allá. Estas imperfecciones son raras y poco perceptibles, pero al realizar la detección de regiones parece que las detecta como regiones de texto y crear la triangulación de Delaunay genera múltiples conexiones adicionales que no deberían existir y que probablemente confundan al modelo en el aprendizaje. Esto se podría arreglar también por ejemplo recortando los bordes más externos de las imágenes.

Otra manera de solucionarlo quizás sería con un algoritmo de separación texto-gráficos más eficiente que eliminase ruido como ese más eficazmente. Examinando las imágenes de texto después de la separación se hace evidente que, aunque sí que elimina buena parte de las componentes gráficas y del ruido, no consigue hacerlo completamente, y estos elementos que quedan después son erróneamente identificados como regiones de texto. Existe una clase en estas, la de no-texto, predicciones específicamente para sortear esta clase de regiones de texto falsas, y aunque no sé hasta qué punto es posible, idealmente solo se deberían identificar regiones de texto, eliminando así la necesidad de esta categoría y convirtiéndose este problema en un problema de clasificación binaria, reduciendo así considerablemente el grado de complejidad del mismo.

Otra forma de potencial mejora es cambiar la manera de abordar la identificación de las regiones de texto. Para el proyecto he utilizado la función que proporciona la librería *scikit-image*, pero quizás haya otras formas de detectar áreas de interés que sean más adecuadas para el problema de identificación de texto. De esta manera se solucionaría también el problema de la detección de ruido, y las regiones de texto se ajustarían mejor a como son realmente.

Relacionado con esto, fruto de estas regiones de texto también se extraen una serie de características de ellas, tal y como se describe en el apartado 3.3 de este documento. No obstante, existen más propiedades que las que he utilizado, y sería interesante también explorar si estas pueden aportar alguna mejora en las predicciones, todavía más si cabe si se utilizase un nuevo método de identificación de regiones, ya que los valores cambiarían de uno a otro incluso para una misma característica. Lo mismo sucede con las propiedades de las aristas. Para el propósito de este proyecto solo se ha utilizado la longitud de las mismas, pero la búsqueda y adición de nuevas propiedades podría fortalecer el aprendizaje del modelo y mejorar su precisión.

En lo referente a la configuración en si del modelo, se ha jugado con los parámetros principales y que más deberían afectar a su funcionamiento. No obstante, todavía cabría realizar más pruebas con ellos, y además hay más opciones de configuración que no se han explorado de igual manera. Si bien a priori no deberían afectar en exceso a los resultados, resulta interesante igualmente experimentar con ellos y ver qué sucede. Por ejemplo, probar pesos distintos en la ponderación de las clases, para intentar prevenir este sesgo que sucede, o modificar manualmente algunos de los parámetros cuyo valor se ha dejado al de por defecto. De manera similar, se han probado 4 valores en escala logarítmica para el parámetro de regularización, pero viendo que este afecta el funcionamiento del programa especialmente cuando sus valores son más altos o bajos, sería interesante ver qué sucede cuando se aplican con cifras todavía más extremas.

Por último, como explicaba anteriormente, el método *lp* es demasiado costoso para mí como para probarlo como es debido, pero el hecho de que con unas pruebas tan parciales ya esté dando algunos resultados mejores que los del resto de pruebas completas me parece muy alentador, y encuentro que sería especialmente interesante experimentar más con este método de inferencia y ver qué resultados daría para unas pruebas completas.

Bibliografía y referencias

[1] Yefeng Zheng, Huiping Li and David Doermann, Member. "Machine Printed Text and Handwriting Identification in Noisy Document Images" *Ieee Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, No. 3, March 2004*

[2] Karl Tombre, Salvatore Tabbone, Loïc PØlissier, Bart Lamiroy, and Philippe Dosch "Text/Graphics Separation Revisited" *LORIA, B.P. 239, 54506 Vandœuvre-lŁs-Nancy, France*

[3] Andreas Mueller. "Semantic Image Segmentation on Pascal VOC" http://pystruct.github.io/auto_examples/image_segmentation.html#semantic-image-segmentation-on-pascal-voc. 2013.

[4] Andreas C. Mueller, Sven Behnke "PyStruct - Structured prediction in Python" *Journal of machine learning, 2014.*

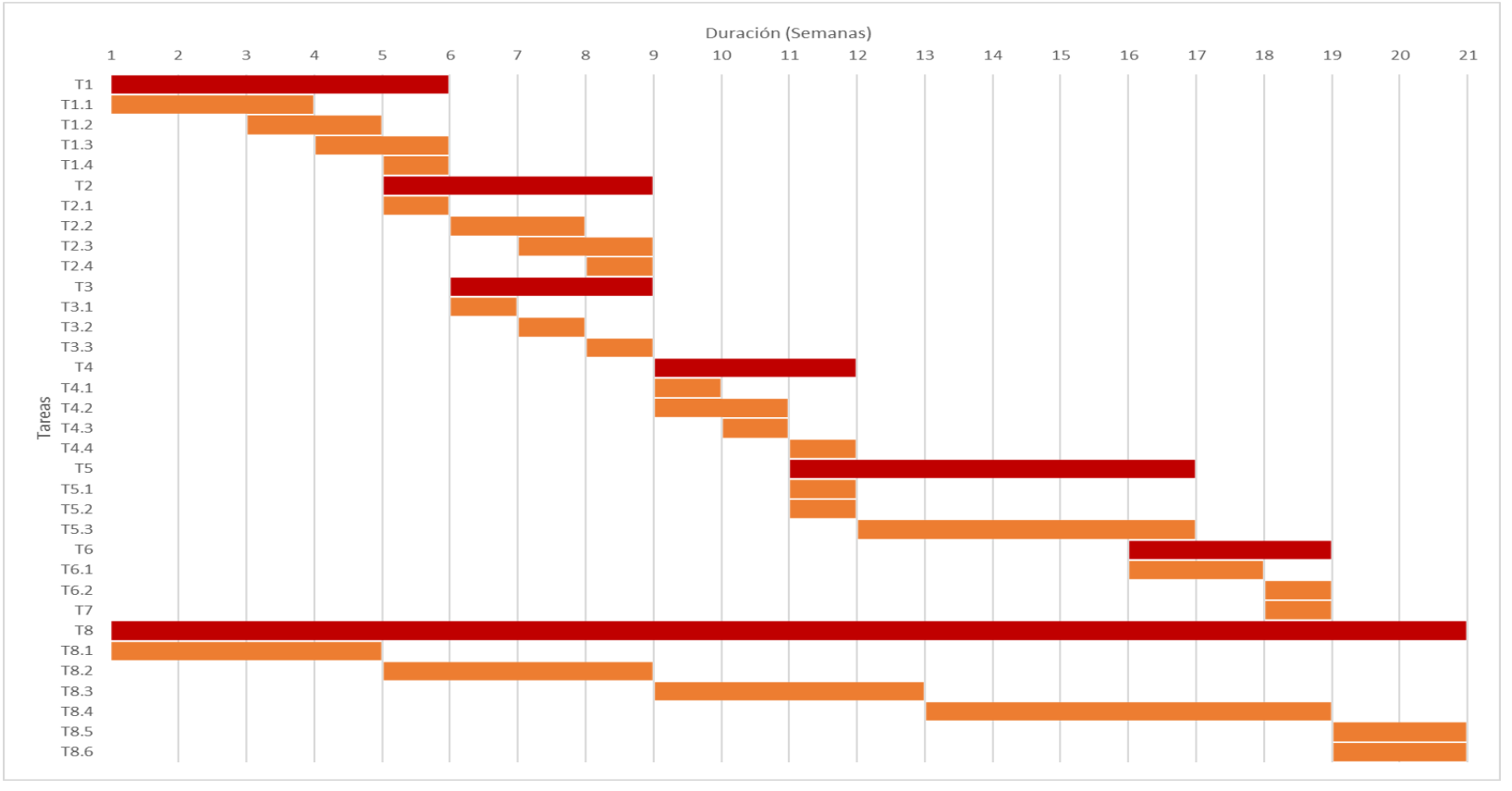
[5] Martin S. Andersen, Joachim Dahl, and Lieven Vandenberghe "CVXOPT" http://cvxopt.org/. 2004-2016.

APÉNDICES

A1. Distribución de tareas

Id Tarea	Nombre Tarea
T1	**Eliminación de ruido y elementos gráficos de la imagen (Bloque C++)**
T1.1	Instalación de programa y librerías necesarias (Visual Studio, OpenCV)
T1.2	Resolver compatibilidades con el código original y compilarlo
T1.3	Pruebas para asegurar el funcionamiento correcto del programa
T1.4	Obtener imágenes conteniendo únicamente la parte de texto de los documentos
T2	**Extracción de características de la imagen (Bloque Matlab, Obsoleto)**
T2.1	Investigar sobre los algoritmos de extracción de características
T2.2	Aplicar la extracción de características a la imagen del texto
T2.3	Generación del grafo de adyacencia de los elementos de la imagen (triangulación de Delaunay)
T2.4	Almacenamiento de los datos extraídos anteriormente en un formato adecuado
T3	**Adecuación de datos y programas para la posterior ejecución de pruebas**
T3.1	Preparar el código de C++ para poder procesar cantidades variables de imágenes en sucesión.
T3.2	Preparar el código de Matlab para poder procesar cantidades variables de imágenes en sucesión.
T3.3	Generar imágenes de Groundtruth a partir de las imágenes y clasificaciones originales
T4	**Extracción de características de la imagen (adaptación a Python)**
T4.1	Investigar sobre los algoritmos de extracción de características
T4.2	Aplicar la extracción de características a la imagen del texto
T4.3	Generación del grafo de adyacencia de los elementos de la imagen (triangulación de Delaunay)
T4.4	Almacenamiento de los datos extraídos anteriormente en un formato adecuado
T5	**Generación del modelo y pruebas de predicción**
T5.1	Investigar funcionamiento de SSVM y las librerías que lo implementan
T5.2	Cargar los datos generados en los pasos anteriores
T5.3	Introducir los datos en el algoritmo de SSVM y generar el modelo
T6	**Comparativa con otras configuraciones y estudio de resultados**
T6.1	Pruebas con diferentes datos y configuraciones y obtención de resultados
T6.2	Comparativa de los resultados obtenidos en las diferentes pruebas (graficas, conclusiones, análisis de datos)
T7	**Examinar resultados del proyecto, y extracción de conclusiones**
T8	**Redacción de documentos**
T8.1	Informe Inicial
T8.2	Informe Seguimiento 1
T8.3	Informe Seguimiento 2
T8.4	Informe Final
T8.5	Diapositivas Presentación
T8.6	Dossier TFG

A1.1 Tabla con la descripción de las tareas y sub-tareas seguidas a lo largo de este proyecto, tal y como han sido planeadas y realizadas



A1.2 Diagrama de Gantt, con la disposición temporal de la realización de las tareas y sub-tareas a lo largo de la duración del TFG. Para saber a qué tarea en concreto corresponde la enumeración, consultar A1.1. Las barras en granate se corresponden a tareas, y las barras en naranja se corresponden a sub-tareas.

A2. Resultados de Experimentos (media de resultados por prueba y categoría)