Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

# Clasificación de correo electrónico mediante computación evolutiva y aprendizaje automatizado

*Email clustering through evolutionary computation and machine learning*

Alberto González Álvarez

La Laguna, 5 de julio de 2019

Dña. **Coromoto León Hernández**, con N.I.F. 78605216W, Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Eduardo Manuel Segredo González**, con N.I.F. 78564242Z, profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

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

Que la presente memoria titulada:

"*Clasificación de correo electrónico mediante computación evolutiva y aprendizaje automatizado*"

ha sido realizada bajo su dirección por D. **Alberto González Álvarez**, con N.I.F. 54115406V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de julio de 2019

# Agradecimientos

*A mis tutores por haber ofertado este Trabajo de Fin de Grado tan interesante y a la vez complejo. Por ayudarme en los momentos más complicados y darme diferentes caminos para resolver los problemas.*

*A mi familia, por el cariño, apoyo y amor con el que he contado durante mi vida. Gracias a el he llegado hasta aquí, y espero seguir llegando aún más lejos. Gracias papi, mami y Anita.*

*Por el amor y el cariño de mi novia, que me ha ayudado cuando las cosas no me han salido como he querido así como también relativizar los problemas, convirtiendo los problemas grandes en pequeños.*

*Por el apoyo y las buenas amistades que me llevo de mi paso por la carrera, sin duda hubiera sido muy diferente si no hubiera conocido a Daute, Juan Pablo, Carlos, Cristian y Angel, sin duda grandes compañeros y grandes amigos que espero seguir manteniendo.*

# Licencia

**Resumen**

*El ingente número de correos electrónicos que puede recibir diariamente una institución puede ser abrumador. La dificultad para determinar los límites que separan a las diferentes temáticas de cada uno de estos correos es un problema de clasificación de texto. En la actualidad se usan técnicas de procesamiento de lenguaje natural, y técnicas de aprendizaje automatizado para tratar de resolverlo.*

*Este Trabajo de Fin de Grado trata de mezclar estas técnicas con otras relacionadas con la inteligencia computacional como son los algoritmos evolutivos. Nos valdremos de ellos para generar diferentes soluciones iniciales que irán evolucionando con el paso de las generaciones, tratando de buscar una solución óptima para el problema, que nos permita, en última instancia, clasificar de forma efectiva un conjunto de correos electrónicos de validación.*

*Para ello se usarán diferentes clasificadores para evaluar la calidad de cada una de las soluciones proporcionadas por el algoritmo evolutivo.*

**Palabras clave:** aprendizaje automatizado, procesamiento de lenguaje natural, inteligencia artificial, algoritmos evolutivos, algoritmos genéticos, clasificación de texto

**Abstract**

*The number of daily emails that an institution receives could be overwhelming. The difficult to decide what email belongs to what category is a text classification problem. Currently, there are a lot of techniques related to Artificial Intelligence (AI) that tries to find a solution for the above problem, for example, natural language processing or machine learning.*

*This project combines the previous techniques with evolutionary algorithms. There will be multiple initial solutions for our problem, and these solutions will be improved in each generation of a genetic algorithm, trying to achieve an optimal solution considering a particular set of categories to classify email..*

*For that purpose, we will use different classifiers to evaluate the quality of the extracted solutions provided by the evolutionary approach..*

**Keywords:** machine learning, natural language processing, artificial intelligence, evolutionary algorithms, genetic algorithms, text classification

# Índice general

# Índice de Figuras

# Índice de Tablas

# Capítulo 1

# Introducción

El término *"Business Intelligence"* surge en la década de los 90 y recoge *“todas las técnicas, tecnologías, sistemas, prácticas, metodologías y aplicaciones que permiten a las empresas a analizar los datos importantes de su negocio, ayudando a entender mejor el modelo de negocio y los mercados a los que se dirige, haciendo posible que se tomen las decisiones de negocio a tiempo”* [1].

La clasificación de textos es un problema clásico que se puede enmarcar dentro de las estrategias de *"Business Intelligence"*. Este trata de determinar a qué categoría puede pertenecer un determinado texto o documento haciendo un análisis de su contenido. Pero no solo eso, en muchas de las ocasiones se realiza un examen detallado del texto, extrayendo información que puede ayudar a su clasificación. Por ejemplo, si busca clasificar noticias deportivas y económicas, resultaría interesante saber a qué diario pertenece cada una de ellas, pues se obtendría una mayor precisión teniendo en cuenta esta variable.

La clasificación de correos electrónicos es una tarea que comparte muchas similitudes con la clasificación de textos pues, al fin y al cabo, cuando se recibe un correo electrónico, lo que tratamos de averiguar es si pertenece a una categoría o a otra teniendo en cuenta el contenido del mensaje, aunque se pueden tener en cuenta muchos otros metadatos. Examinar estos metadatos puede resultar especialmente interesante, ya que, en la mayoría de ocasiones, los correos que provienen de una misma dirección tienen una mayor probabilidad de tratar la misma temática que correos anteriores provenientes de la misma dirección. Sin duda poder automatizar la tarea de clasificación resulta especialmente interesante, pues a diario, *“un trabajador de oficina recibe una media de 121 correos”* [2].

En la mayoría de los casos suelen haber sistemas que resuelven este problema, determinando si un correo debe ir a un departamento u otro con un conjunto de reglas sencillas, que, en cualquier caso, deben ser establecidas manualmente. Por ejemplo, si el correo entrante contiene en el asunto la palabra $X$, entonces redirigirlo al departamento de finanzas, o, si el correo entrante proviene de la dirección $Y$, entonces redirigirlo al departamento de marketing. Esto, en la mayoría de las ocasiones, resulta una tarea compleja ya que requiere estar continuamente actualizando el sistema de reglas de una forma manual.

Sin embargo hay técnicas mucho más avanzadas para determinar a qué categoría pertenece un email o un texto. Por ejemplo, en la literatura encontramos problemas similares que se han resuelto haciendo uso de redes neuronales convolucionales [3], usando *Support Vector Machine* (SVM) [4] o también algoritmos evolutivos [5], que será

la técnica que se empleará en este Trabajo de Fin de Grado. Todos ellos se basan en el procesamiento de lenguaje natural, el cual describiremos en el siguiente apartado.

## 1.1. Procesamiento de lenguaje natural

Antes de definir en qué consiste el procesamiento de lenguaje natural, definiremos lenguaje natural y lenguaje formal [6][7].

- **Lenguaje formal**. Es el lenguaje empleado para transmitir conocimiento científico. Por ejemplo, los problemas matemáticos se expresan a través de este lenguaje cuando se trata, por ejemplo, de una demostración formal. Sin embargo, no es el único. Los lenguajes de programación también pertenecen a esta categoría. Ambos se desarrollan a partir de una gramática predefinida y no hay ambigüedad, razón por la cual se usan en el ámbito científico.
- **Lenguaje natural**. Al contrario que los lenguajes formales, los lenguajes naturales son empleados para la comunicación entre personas, y por tanto, pueden dar cabida a la ambigüedad. Todos los idiomas forman parte de este conjunto.

Una vez que entendemos la diferencia entre los dos tipos de lenguajes introduciremos el concepto de procesamiento de lenguaje natural. El procesamiento de lenguaje natural o también conocido como *Natural Language Processing* (NLP) está relacionado con los campos de la lingüística y la inteligencia artificial, pues el fin último es tratar de descifrar el significado de un mensaje teniendo en cuenta que el lenguaje natural, por definición, es ambiguo, y que por tanto cobra importancia el contexto en el que se encuentre un mensaje. Por ejemplo, imaginemos que tenemos estas dos oraciones:

*"El domingo pasaré a votar en el colegio electoral"*
y
*"El domingo paso de votar en el colegio electoral"*

Es evidente que el significado de ambas es radicalmente distinto. En la primera de ellas se manifiesta la intención de ir a votar, mientras en la segunda se puede observar que el sujeto no va a ir a votar. Sin embargo, la composición de palabras es prácticamente igual, variando tan solo en la preposición, pero esto hace que cambie radicalmente el sentido de la oración. Estos casos son realmente complicados tratar y diferenciar, pero con el uso de algunas técnicas como los n-gramas se pueden tratar de forma más o menos efectiva [8].

## 1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado (TFG) es conseguir clasificar un conjunto de correos electrónicos de entrada en diferentes categorías según su contenido. Esto se intentará conseguir reduciendo el número de palabras empleadas para representar a cada una de las categorías, teniendo en cuenta que cada categoría contiene un conjunto de palabras.

Con el fin de alcanzar el objetivo de este TFG, se necesitará un conjunto de validación que evalúe la calidad del modelo a la hora de diferenciar entre las categorías. Además,

también se requerirá un conjunto de entrenamiento del cuál se extraigan las palabras pertenecientes a cada categoría y con el que entrenaremos nuestro algoritmo genético.

Para desarrollar el TFG se han valorado las diferentes técnicas mencionadas anteriormente, eligiendo finalmente, trabajar con Algoritmos Evolutivos [5], pues en muchas de las ocasiones se obtiene un resultado de forma rápida y con una precisión más que aceptable. Además, en la literatura no se encuentran demasiados problemas abordados con esta técnica, por lo que nos resultó interesante probar su efectividad.

Los objetivos que se pretenden alcanzar son los siguientes:

1. Clasificar los correos electrónicos según su categoría o la temática del mismo intentando obtener los mejores resultados. Al mismo tiempo, se desea reducir la cantidad de palabras necesarias para ello.
2. Conseguir aplicar diferentes técnicas de procesamiento para preparar los datos de entrada, eliminando aquellos que son irrelevantes, pues estos son casi tan importantes como el sistema clasificador.
3. Probar y experimentar con algoritmos evolutivos, pues la gran parte de los problemas de este estilo no se suelen resolver usando esta técnica.
4. Trabajar con técnicas de aprendizaje supervisado.
5. Resolver un problema real planteado por una consultoría que estaba interesada en clasificar su correo electrónico.
6. Desarrollar una interfaz gráfica sencilla que muestre la evolución de la población en cada generación, pudiendo observar la mejora entre una generación y otra.
7. Generar código reutilizable y modular para otros problemas relacionados con algoritmos genéticos.

## 1.3. Metodología y Plan de Trabajo

Las tareas a ejecutar para el desarrollo de este Trabajo de Fin de Grado son las siguientes:

- Tarea 1. Revisión bibliográfica. Formación y estado del arte.
- Tarea 2. Obtención de datos.
- Tarea 3. Diseño del sistema de clasificación.
- Tarea 4. Implementación del sistema.
- Tarea 5. Validación y evaluación del sistema.
- Tarea 6. Redacción de la presente memoria y análisis de resultados.

El TFG estaba pensado para colaborar con una consultoría. Esta se encargaría de proporcionarnos un conjunto de correos etiquetados y en español. Con estos datos, nosotros tendríamos que desarrollar una aplicación que permitiera clasificar nuevos correos entrantes a partir de la información extraída de los datos de entrenamiento.

Sin embargo, la consultoría finalmente no nos facilitó estos datos, complicándose considerablemente el desarrollo del trabajo. Ante esta situación se plantearon dos escenarios:

- Trabajar con un conjunto de datos similar y etiquetados en inglés.
- Trabajar con un conjunto de datos similar, no etiquetados, y en español. Pues no se encontraron datos similares en inglés que estuvieran etiquetados.

Ante estas dos opciones se eligió la primera de ellas, pues está más relacionada con el planteamiento inicial de este TFG a pesar de que los datos fueran en inglés.

En los siguientes capítulos se hablará de los antecedentes y el estado actual del tema. A continuación, se hablará acerca de los fundamentos teóricos en los que se basa el desarrollo de este TFG, seguido por la formalización e implementación de la aplicación. En los últimos apartados se hablará sobre los resultados obtenidos, las conclusiones extraídas y potenciales líneas de trabajo futuro y por último se elaborará un presupuesto para la implementación del sistema.

# Capítulo 2

# Antecedentes y estado actual

Existen múltiples técnicas que pretenden aportar soluciones al problema de la clasificación de textos. La implementación de cada una de ellas es diferente, sin embargo, todas pretenden producir un modelo o solución con las palabras que mejor representen a cada categoría.

En este capítulo se hablará sobre las diversas técnicas usadas para abordar el problema y se distinguirán según su grado de facilidad a la hora de usarlas e implementarlas. Según estos dos principios, podemos dividir el capítulo en técnicas sencillas y técnicas complejas.

Por último, se contrastará brevemente con la técnica propuesta en este TFG.

## 2.1. Técnicas sencillas

Las técnicas sencillas son aquellas que están al alcance de cualquier usuario medio, por ejemplo, los típicos filtros de servicios de correo electrónico como *Gmail* o *Outlook*. Estos filtros se engloban dentro de las técnicas sencillas, pues hay que añadir manualmente cada una de las reglas, y además, no hay un aprendizaje detrás sino que se basan en la aplicación de las reglas definidas con el criterio del usuario.

Las reglas que se pueden definir en los servicios de correo electrónico clásicos permiten especificar las condiciones que se deben dar para considerar que un correo pertenece a una categoría o a otra. Los proveedores de servicios de correo electrónico permiten, por ejemplo, definir condiciones tales como *"Si el correo llega de la dirección buy@amazon.es y el mensaje contiene las palabras compra, zapatos, ropa, accesorio, tecnología entonces es un email que pertenece a la categoría de compras"*, Figura 2.1.

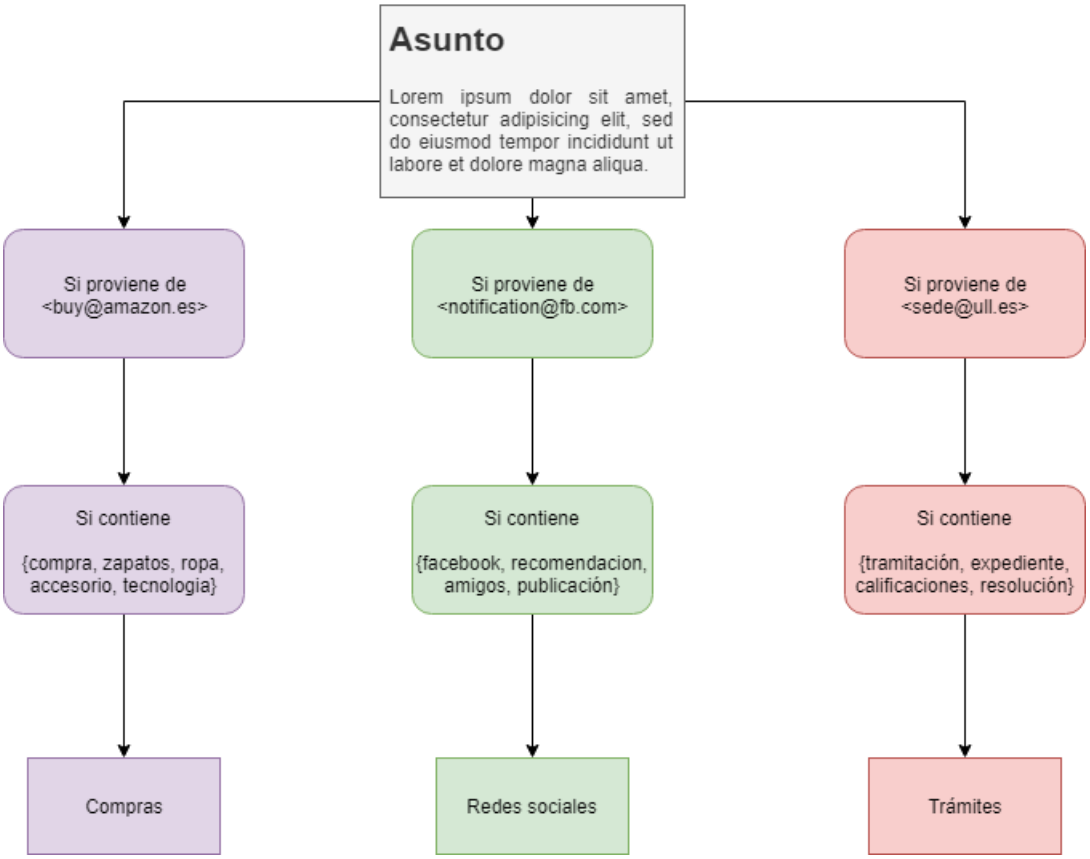


Figura 2.1: Diagrama de funcionamiento de un sistema de clasificación sencillo.

Estas reglas permiten una clasificación básica de correos. Sin embargo existe la necesidad de elaborar modelos más complejos que sean capaces de extraer el conocimiento de los datos sin tener que estar definiendo cada una de las reglas manualmente.

## 2.2. Técnicas complejas

Las técnicas complejas, normalmente, no se encuentran accesibles para un usuario medio. Aunque sí existen algunas soluciones comerciales enfocadas al sector empresarial como *Boldon James* [9], que entre otras cuestiones, permite clasificar correos .

Algunas de las técnicas empleadas para la resolución de problemas de clasificación de texto son SVM [4], o algoritmos evolutivos [5].

### 2.2.1. Basadas en SVM

Hoy en día la técnica más utilizada para clasificar textos es el uso de SVM. Este modelo de aprendizaje supervisado consiste en dividir un espacio multidimensional de tal forma que los puntos que representan palabras o características de una categoría queden separados por un hiperplano de una dimensión menor a la del espacio multidimensional representado. En la clasificación de textos, cada una de estas dimensiones representaría a una categoría diferente y cada punto en el espacio representa una palabra. Por tanto, si tenemos dos categorías tendremos dos dimensiones y el hiperplano que dividirá estas dos categorías, será una línea que separará ambas categorías. Pongamos un ejemplo sencillo:

Imaginemos que nos interesa clasificar textos o documentos en función de si pertenecen a una categoría de crímenes ($C_{Crimen}$) o de economía ($C_{Economía}$). Para saber si es de una categoría o de otra, necesitamos conocer el vocabulario característico de cada una de ellas. Por ejemplo, el vocabulario podría ser el que aparece en 2.1.

$$\begin{aligned} V_{Economía} &= \{dólar, producto, vender, silla\} \\ V_{Crimen} &= \{arma, pistola, cuchillo, blanca\} \end{aligned} \tag{2.1}$$

Una vez tenemos el vocabulario necesitamos entrenar el modelo, para obtener nuestro hiper-plano que luego permitirá clasificar a nuevos documentos.

Vamos a poner de ejemplo las oraciones que están en (2.2) y en (2.3). Está claro que a simple vista hay verbos que se encuentran conjugados en las citadas oraciones, y que por tanto, aunque sea el mismo verbo, al encontrarse conjugado en diferentes personas, se tratará como si fueran palabras totalmente diferentes.

$$\begin{aligned} D_{Economía} = \{&\text{Los productos de allí son mejores,} \\ &\text{Las armas se están vendiendo por un dólar}\} \end{aligned} \tag{2.2}$$

$$\begin{aligned} D_{Crimen} = \{&\text{La pistola es el arma más común,} \\ &\text{El cuchillo es un arma blanca}\} \end{aligned} \tag{2.3}$$

Para evitar el problema anteriormente mencionado, haremos un preprocesamiento de los datos, el cual podremos ver en la Sección 5.1. Una vez aplicado, el resultado las oraciones resultantes serían las que figuran en (2.4) y en (2.5).

$$\begin{aligned} D_{Economía} &: \{d_{e1}, d_{e2}\}, \quad |D| = 2 \\ d_{e1} &: \textit{“producto allí mejores”} \\ d_{e2} &: \textit{“arma estar vender dólar”} \end{aligned} \tag{2.4}$$

$$\begin{aligned} D_{Crimen} &: \{d_{c1}, d_{c2}\}, \quad |D| = 2 \\ d_{c1} &: \textit{“pistola ser arma común”} \\ d_{c2} &: \textit{“cuchillo ser arma blanca”} \end{aligned} \tag{2.5}$$

Como se puede observar, cada documento está formado por un conjunto de palabras a las que idenfiticaremos como $w_i$ siendo $i$ el índice de la palabra en un documento dado.

A continuación contaremos la frecuencia con la que aparecen términos que están, al menos, en alguno de los vocabularios de las diferentes categorías.

$$\begin{aligned} &(\{w_1\} \in V_{Economía}) \subset d_{e1} \\ &(\{w_3, w_4\} \in V_{Economía} \wedge \{w_1\} \in V_{Crimen}) \subset d_{e2} \\ &(\{w_1, w_3\} \in V_{Crimen}) \subset d_{c1} \\ &(\{w_1, w_3, w_4\} \in V_{Crimen}) \subset d_{c2} \end{aligned} \tag{2.6}$$

Si representáramos esta información en el espacio multidimensional que previamente mencionábamos, el resultado sería similar al que se puede apreciar en la Figura 2.2. Vemos que el documento $d_{e2}$ está más separado de la categoría de economía porque tiene una palabra que no está presente el el vocabulario de esa categoría. Véase (2.6).

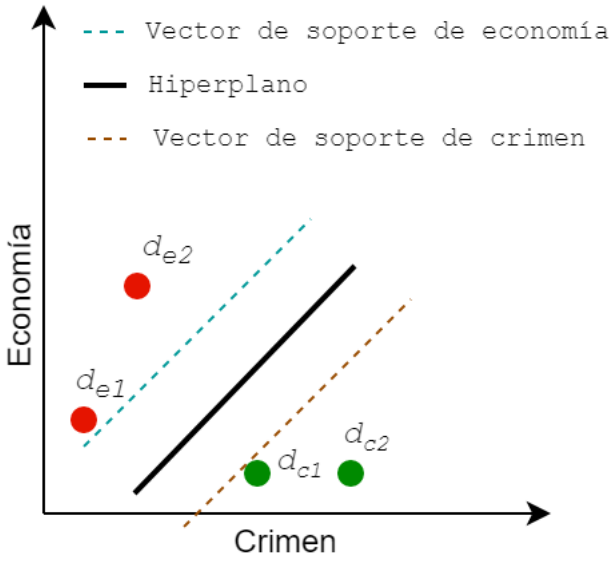


Figura 2.2: Gráfico que representa los documentos según su categoría.

La línea continua que se puede visualizar en el gráfico es el hiperplano del que se hablaba previamente. Este hiperplano se colocará de tal forma que se diferencien lo mejor posible los puntos de una categoría y los de otra. Para ello, podríamos tomar como puntos de referencia $d_{e1}$ y $d_{c1}$, pues son los que siempre quedarán más cercanos a este hiperplano, y, por tanto, los que más nos interesa diferenciar. Los vectores que limitan a cada una de las categorías se conocen por el nombre de Vector de soporte, de ahí el nombre de esta técnica. Cualquier punto que no quede entre los dos vectores de soporte podrá ser clasificado con éxito. Si hubiera uno en medio no se podría determinar con exactitud la categoría a la que pertenece.

Cabe destacar que este es un caso trivial, pues hay ocasiones en las que la separación no es tan sencilla, pudiéndose producir otro tipo de figuras diferentes. Por ejemplo, como la que se puede apreciar en la Figura 2.3.

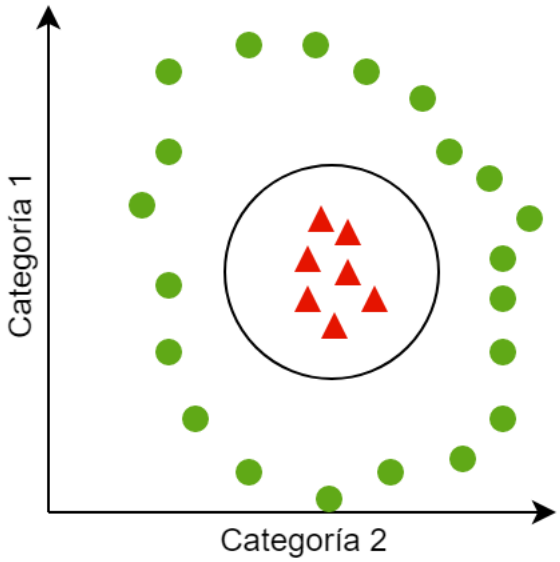


Figura 2.3: Gráfico que representa características de los datos según la categoría.

### 2.2.2. Basadas en computación evolutiva

Como se mencionaba con anterioridad, actualmente hay una gran variedad de problemas resueltos con esta técnica. Sin embargo ha resultado algo más complicado encontrar proyectos en los que se usara para resolver un problema de clasificación de texto.

A pesar de ello, si se encontró un trabajo previo en el que tratan de clasificar un correo según se considere *"spam"* o *"ham"* [5]. Los correos *"spam"* son, normalmente, correos no deseados que se envía con fines publicitarios. Por su parte, los correos *"ham"* son correos que no son *"spam"*, y por tanto, son correos "deseados". El problema es, por tanto, un problema de clasificación de textos, donde se pretende detectar expresiones en los correos que puedan proporcionar los suficientes indicios como para clasificar a ese correo como *"ham"* o *"spam"*.

Para resolver el problema, se usan algoritmos evolutivos. Como entrada a este algoritmo, tenemos un corpus de correos que son *"spam"* y otro corpus de correos *"ham"*, de tal forma que se analiza cada una de las líneas de *"spam"*, generando una expresión regular a partir de ellas. Una vez obtenida, y para evaluar si efectivamente esa expresión sirve para detectar correos *"spam"*, se consulta el corpus de correos *"ham"*. Si la expresión regular no coincide con ningunos de los correos, entonces se puede afirmar que se trata de una expresión regular que no clasifica bien a los correos *"ham"*, y, por el contrario, sí que clasificaría bien a un correo *"spam"*. Por tanto, los individuos de la población estan conformados por expresiones regulares válidas para detectar patrones de spam en correos, que es el objetivo que se perseguía.

El proceso de evaluación se hace teniendo en cuenta el número de correos que contienen al menos un patrón *"spam"*. Para evitar expresiones regulares excesivamente ajustadas al conjunto de entrenamiento, se penalizarán aquellas expresiones regulares que tengan una mayor longitud.

La intención de explicar este trabajo previo es la de establecer un punto de referencia con el TFG desarrollado. En el siguiente apartado se explicará brevemente cómo se ha

abordado este TFG, y en capítulos posteriores se detallará todo el trabajo realizado y su implementación.

## 2.3. Solución propuesta

Este trabajo se ha enfocado para resolver el problema de clasificación de correos electrónicos haciendo uso de algoritmos evolutivos, más concretamente, algoritmos genéticos. Creemos que tienen el suficiente potencial para hallar buenas soluciones a la vez que enfocamos el problema de una manera novedosa. En el proyecto se abarca todo el proceso, es decir, desde la extracción de los datos, pasando por el procesamiento de los mismos, hasta al final obtener solución y su evaluación.

Por tanto el objetivo de este TFG es reducir el conjunto de palabras necesarias para determinar a qué categoría pertenece un correo electrónico y una vez obtenido este conjunto de palabras, clasificar una serie de correos con un conjunto de datos diferente. Para calcular la función fitness o de evaluación de cada solución, se ha usado un clasificador. Más adelante detallaremos la implementación de esta función de evaluación.

# Capítulo 3

# Fundamentos teóricos

En el capítulo anterior se ha descrito el objetivo que se persigue en este trabajo y alguna de las técnicas que se usan actualmente para tratar de resolver estos problemas. Sin embargo, como usaremos computación evolutiva para resolverlo, dedicaremos un apartado a la computación evolutiva.

## 3.1. Computación evolutiva

La computación evolutiva surge en la década de los sesenta del siglo pasado. Fue resultado del estudio de John Holland [10][11], quien propuso utilizar técnicas de selección y supervivencia para resolver problemas que ya estaban resueltos por la propia naturaleza, pero que resultaban ser intratables si se intentaban resolver con un ordenador de la época. En un principio, sus ideas no sirvieron para abordar problemas muy complejos, pues en ese entonces el poder de cómputo de los ordenadores era escaso. Sin embargo, a partir de 1985, la aparición de computadores de altas prestaciones y la reducción de los costes de producción, hicieron posible que se empezaran a utilizar estas técnicas.

La computación evolutiva se basa en los procesos evolutivos que tienen lugar en la naturaleza, pero enmarcados en el mundo de la computación. Por definición, son algoritmos no deterministas o estocásticos, pues no siempre se obtiene una salida igual ante una misma entrada. Nos encontramos por tanto, ante algoritmos heurísticos, los cuales tratan de buscar soluciones óptimas a un problema sin evaluar todo el conjunto de posibles soluciones, véase la Figura 3.1. Esto evidentemente es lo que hace que el tiempo para obtener una buena solución se vea reducido drásticamente.

Sin embargo, la función de evaluación elegida para el problema que se vaya a tratar influirá en el tiempo de cómputo del algoritmo, pues esta función, como veremos más adelante, se ejecuta numerosas veces.

La principal ventaja de los algoritmos evolutivos reside en que se puede resolver un problema sin tener una base matemática sólida, además, normalmente no hace falta adaptar el diseño de un algoritmo evolutivo a un problema. Sin embargo, si se precisa encontrar una forma óptima para evaluar la calidad de las soluciones, y para ello, si es necesario conocer bien el problema al que nos enfrentamos.

Dentro de la computación evolutiva o algoritmos evolutivos, se encuentran múltiples paradigmas según su funcionamiento, pero nos centraremos en el funcionamiento de los algoritmos genéticos por ser los que se han usado para el desarrollo de este trabajo.

Generalmente, los algoritmos genéticos se encuentran compuestos por: una población, operadores genéticos (cruce y mutación, generalmente), operadores de reemplazo, función de evaluación o *fitness* y operador de selección de padres.

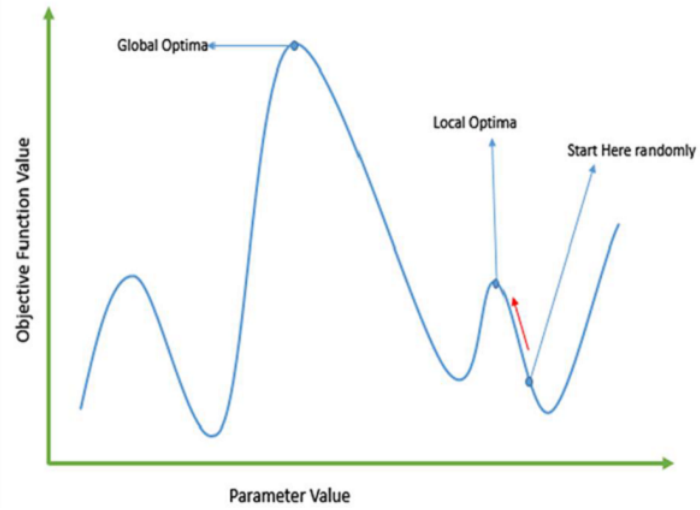


Figura 3.1: Representación de las soluciones según los parámetros de entrada.

### 3.1.1. Algoritmos genéticos

Si bien la computación evolutiva surge en la década de los sesenta, no es hasta 1975 que John Holland introduce los algoritmos genéticos. Un algoritmo genético siempre se estará compuesto por los siguientes componentes: la población, los individuos, la función de evaluación o fitness, los operadores evolutivos, los operadores de reemplazo y la condición de parada. Estos componentes serán pormenorizados a continuación.

- **Población**: La población se encuentra compuesta por un conjunto de individuos.
- **Individuo**: Cada individuo de la población representa una solución válida del problema. La información que contiene el individuo es almacenada en su cromosoma y cada una de las unidades que compone el cromosoma se denomina gen. En un inicio, estas unidades eran normalmente binarias (0's y 1's).
- **Función de evaluación o fitness**: Es un cálculo numérico o puntuación que se asocia a cada individuo e indica su calidad. Esta función de evaluación es la que variará según el problema, aunque siempre se pueden usar algunas que son independientes del mismo. Por ejemplo, se puede usar la precisión que se consigue al clasificar un conjunto de datos.
- **Operadores evolutivos**: Los operadores evolutivos básicos son los operadores de cruce, o recombinación, y el de mutación. El primero de ellos mezcla los cromosomas de dos individuos distintos (denominados padres), produciendo, normalmente, otros dos individuos (hijos). Por su parte, la mutación altera los genes de un cromosoma.
- **Operadores de reemplazo**: Los operadores de reemplazo son los encargados de seleccionar los individuos para la siguiente generación. Se evalúa la calidad de los padres e hijos y se deciden cuáles de ellos serán los que conformen la población en la siguiente generación.

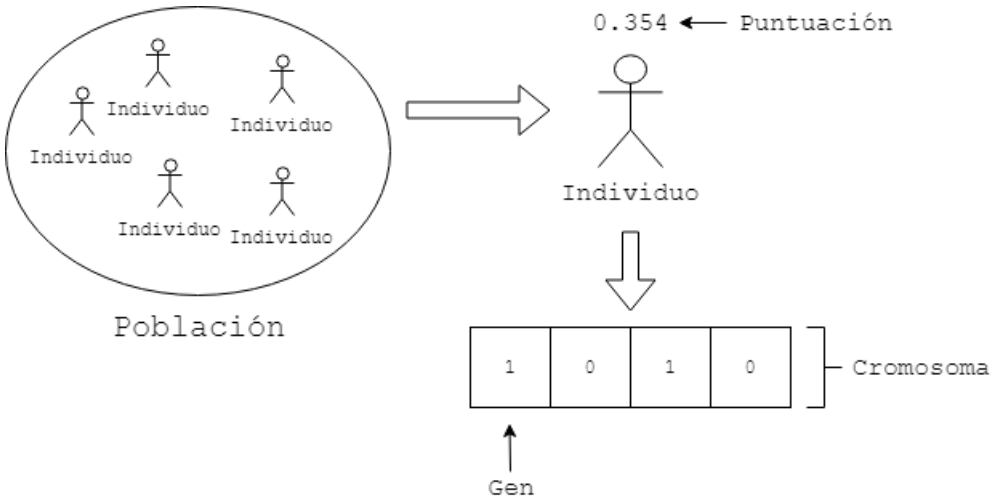


Figura 3.2: Representación de los componentes de un algoritmos genético.

Una vez expuestas las partes por las que se encuentra compuesto un algoritmo genético, explicaremos como es el funcionamiento del mismo.

### 3.1.2. Funcionamiento

A continuación se explicará el funcionamiento general de un algoritmo genético, apoyándonos en el Algoritmo 1.

1. Inicialmente, se genera una población para el problema. Una población es un conjunto de individuos. Cada uno de ellos representa una posible solución a nuestro problema. Esta población se puede generar de dos maneras diferentes: de forma aleatoria o añadiendo conocimiento específico del problema en cuestión.

   Los individuos o soluciones generadas en la población inicial pueden tener gran importancia en aquellas ocasiones en las que el número de iteraciones o de generaciones no es elevado. En estos casos hay una mayor probabilidad de que, esta población inicial, determine las futuras soluciones. Por tanto, generar una población muy mala, puede repercutir en la calidad del resto de soluciones en las siguientes generaciones.

2. Una vez generada esta población se procede a evaluar a cada uno de los individuos, asignándole un valor *fitness*. Este se calculará con la función que recibe el mismo nombre.

3. Mientras la condición de parada del algoritmo no se cumpla se repetirá lo siguiente.

   *a*) Se realiza una selección de dos individuos que serán los que denominaremos padres. Estos padres pueden ser seleccionados siguiendo múltiples estrategias de selección. Entre ellas se encuentran la selección por torneo binario, por sorteo o *roulette wheel* [12].

*b*) Estos dos padres se cruzan, intercambiando parte de la información genética y produciendo, generalmente, dos hijos. Con el fin de introducir nueva información genética que pueda producir, eventualmente, una mejora de la solución se aplica una mutación.

   Existen un sinfín de técnicas para realizar el cruce y la mutación [12].

*c*) Con los operadores de reemplazo decidiremos qué individuos, entre padres e hijos, serán los que conformen la siguiente población. Una vez seleccionados estos individuos, se volverá a repetir todo el proceso. También existen múltiples operadores de reemplazo.

**Algoritmo 1** Pseudocódigo algoritmo genético

```
1:  function algoritmoGenetico
2:      Población ← Inicialización de la población
3:
4:      while CondicionParada == false do
5:          SiguientePoblación ← []
6:          for i = 1 hasta |Población|/2 do
7:              Padre_1, Padre_2 ← Seleccionar prades de Población
8:
9:              Hijo_1, Hijo_2 ← Aplicar operador mezcla(Padre_1, Padre_2)
10:             Hijo_1 ← Aplicar operador mutación(Hijo_1)
11:             Hijo_2 ← Aplicar operador mutación(Hijo_2)
12:
13:             SiguientePoblación ← EstrategiaReemplazo(Padres, Hijos)
14:
15:         Población ← SiguientePoblación
16:     return Población
```

# Capítulo 4

## Obtención de datos

Como se mencionó en la Sección 1.3, el trabajo estaba planificado para que una consultoría nos proporcionara los datos de los correos electrónicos. La intención era aportar una solución al problema que estaban teniendo para diferenciar los correos. Sin embargo, la consultoría no nos llegó a proporcionar nunca estos datos y se optó por buscar un conjunto que tuviera unas características similares. La tarea no fue sencilla, pues teníamos que encontrar un conjunto lo suficientemente amplio como para dividirlo en dos subconjuntos: uno de entrenamiento con el que alimentaríamos al algoritmo genético y otro de validación con el que evaluaríamos la calidad de la solución proporcionada por este algoritmo.

Durante el proceso de búsqueda se encontraron algunos conjuntos de datos en español en diferentes sitios, entre ellos *Kaggle* [13]. Sin embargo, los conjuntos que se encontraron no eran lo suficientemente extensos. Pero el mayor problema no era este, sino que no se encontró un conjunto que estuviera etiquetado, lo cual implica, que no podríamos haber usado técnicas de aprendizaje supervisado.

Sumado a todo esto, los datos encontrados en español, no compartían similitudes con la estructura básica de un correo (email, asunto y mensaje). Por ello, se centraron los esfuerzos en conseguir un conjunto de datos en inglés, pues es más sencillo encontrar datos en este idioma.

Desde el principio se tuvo constancia de los datos que la librería *Sklearn* [14] incluía para hacer pruebas con los diferentes métodos que implementaba. Sin embargo, se evitó usar estos datos porque no tenían tantos metadatos como los que nos proporcionaría la empresa. Ver Apéndice A.1.

El proceso de búsqueda continuó y se encontró otro conjunto de datos que es usado habitualmente en la literatura. Se trata de los correos de la compañía *Enron* [15]. Sin embargo, este conjunto de datos tampoco estaba etiquetado. Hasta ese entonces era el conjunto más parecido que habíamos encontrado, por tanto se optó por buscar este mismo conjunto de datos pero etiquetados. Finalmente se encontraron [16], pero resultaba difícil establecer diferencias entre las categorías debido a la amplía variedad tanto de categorías como de subcategorías empleadas para clasificar estos correos.

Como no se conseguía obtener un conjunto de datos que satisficiera nuestros requisitos, se optó por usar el que habíamos encontrado en el módulo *dataset* de la librería *Sklearn* [14]. La ventaja que tienen estos datos es que son un poco más extensos que los datos

etiquetados de *Enron* y además, sus categorías están claramente diferenciadas. El formato de estos datos es el que se puede ver en el Apéndice A.2.

Las categorías extraídas, y con las que trabajremos son las siguientes:

- **Religión** (soc.religion.christian).
- **Armas** (talk.politics.guns).
- **Oriente Medio** (talk.politics.mideast).

Se eligieron estas categorías porque pueden tener palabras en común que quizás complique la tarea al algoritmo genético, y por tanto, sea más cercano a lo que podríamos encontrar en un caso real.

En el siguiente capítulo se hablará de los dos módulos principales que componen nuestra aplicación: el módulo de análisis de correos y el módulo genético que contiene el algoritmo genético.

# Capítulo 5

# Formalización e implementación

En este capítulo se explicarán cada uno de los módulos de los que se compone la aplicación. A continuación, explicaremos brevemente la función que cumple cada uno e ilustraremos las relaciones que hay entre ellos.

- **Módulo *EmailParser***. Es el nombre que recibe el módulo encargado del preprocesamiento de los correos. También extrae la información más relevante de los metadatos de los correos.
- **Módulo *Utilities***. Así es como se le ha denominado al módulo encargado de realizar operaciones genéricas que son usadas por el resto de los módulos. Es un módulo pequeño en comparación con otros de la aplicación.
- **Módulo *Log***. Se encarga de mostrar los mensajes por pantalla de cualquier evento que suceda en cualquiera de los otro módulos de la aplicación. Ver Figura 5.1. También se trata de un módulo pequeño.
- **Módulo *Genetic***. Alberga el algoritmo genético desarrollado para cumplir el objetivo de este TFG. Cuenta con una gran cantidad de clases para representar a cada uno de los componentes del algoritmo genético.

```
> 23:56:21 [Utilities.FileUtilities][INFO] -> Reading JSON file from 'C:\Users\Alberto\PycharmProjects\TFGCode\Data\train_data.json'
> 23:56:21 [Utilities.FileUtilities][INFO] -> Reading JSON file from 'C:\Users\Alberto\PycharmProjects\TFGCode\Data\test_data.json'
> 23:56:21 [EmailParser.DataCategory][INFO] -> Added new category to train 'soc.religion.christian'.
> 23:56:21 [EmailParser.DataCategory][INFO] -> Added new category to train 'talk.politics.guns'.
> 23:56:21 [EmailParser.DataCategory][INFO] -> Added new category to train 'talk.politics.mideast'.
> 23:56:21 [EmailParser.DataCategory][INFO] -> Added new category to validate 'soc.religion.christian'.
> 23:56:21 [EmailParser.DataCategory][INFO] -> Added new category to validate 'talk.politics.guns'.
> 23:56:21 [EmailParser.DataCategory][INFO] -> Added new category to validate 'talk.politics.mideast'.
> 23:56:23 [Genetic.GeneticAlgorithmSpecification][INFO] -> Problem specification loaded!
> 23:56:23 [Genetic.GeneticAlgorithm][INFO] -> Problem specification loaded, ready to start!
> 23:56:31 [Genetic.GeneticAlgorithm][INFO] -> 0th generation: Best individual => [0.7976928735935892: 10977]
```

Figura 5.1: Captura de pantalla de los mensajes que muestra el módulo *Log*.

A continuación, en la Figura 5.2 ilustramos el funcionamiento general de la aplicación. En las próximas secciones se abordará, con más detalle, la implementación de los dos principales módulos: *EmailParser* y *Genetic*.

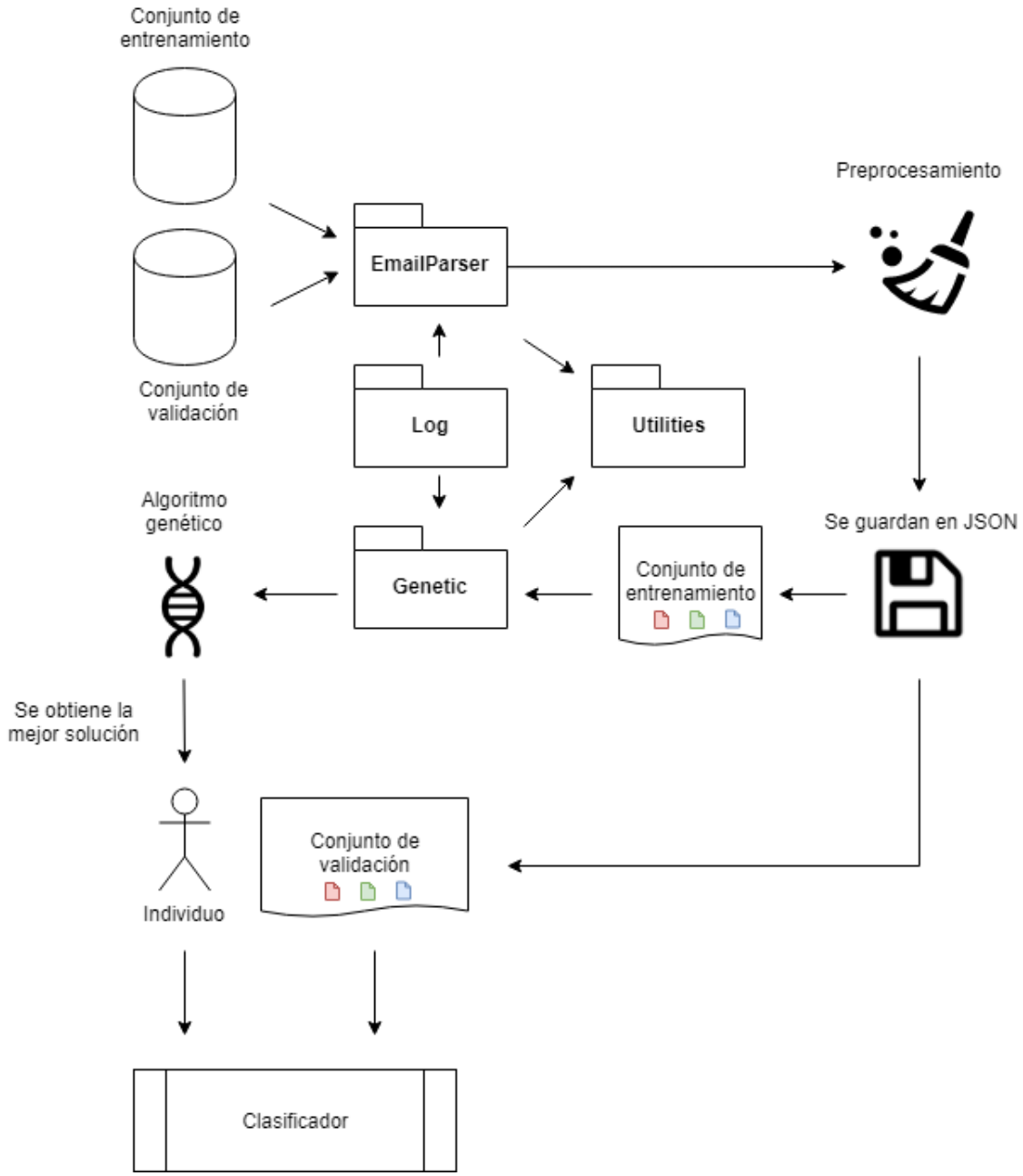


Figura 5.2: Esquema del funcionamiento general de la aplicación.

## 5.1. Módulo *EmailParser*

En esta sección se hablará del módulo *EmailParser*, el cuál se encarga de transformar y procesar los datos de entrada para ser usados por el algoritmo genético. Este procesamiento y transformación se realiza en los datos de entrenamiento pero también en los de validación.

El módulo *EmailParser* procesa todos los correos que descargamos de la librería Sklearn [14] con el método `fetch_20newsgroups()`. Este método se encuentra implementado en el módulo *dataset* de la librería. En un principio, cuando los datos son descargados tienen un formato como el que se aprecia en el Apéndice A.2. Sin embargo, esto no es todo lo flexible que necesitamos, pues nos forzaría siempre a trabajar con contenido que realmente no aporta ningún valor al modelo. Por ejemplo, la información a cerca

de cuantas líneas hay, o las el propio nombre de la etiqueta que indica de qué dirección proviene el correo.

Es por ello que se surge la necesidad de elaborar varias clases que traten todos estos correos para identificar la información relevante y desechar aquella que no nos aporte nada.

Lo primero que hacemos es llamar a la función `parseEmail()` definida en la clase `Email` de nuestra aplicación. Esta función se encarga de identificar los metadatos presentes en los correos, tal y como figura en el Apéndice A.2. La identificación de estos campos se hará con expresiones regulares.

La primera dificultad que encontramos fue identificar la cabecera, pues no todas eran iguales. Había ocasiones en las que algún metadato no estaba y ocasiones en las que sí. Por ello, tuvimos que encontrar algún metadato que estuviera presente todos, y que por tanto, nos permitiera determinar las líneas que ocupaba esta cabecera. Para ello usamos la siguiente expresión regular: `((.|\n)*?Lines:\s*?\d+)`. Una vez identificada, dividimos el correo en 2 secciones: cabecera y contenido.

A continuación, identificamos cada uno de los metadatos que están presentes en la cabecera. Nuevamente se hace uso de expresiones regulares para seleccionar el contenido de cada uno de estos metadatos. Como no todos los correos tienen todos los metadatos, las expresiones regulares debían contemplar que algunos de ellos no aparecieran. Una vez identificados estos datos, aplicamos el procesamiento de palabras únicamente, sobre el campo *Subject*. El campo *Organization* no se usó al no ser un metadato típico en los correos electrónicos. Sin embargo, se puede configurar la aplicación para que también lo tenga en cuenta. Veremos cómo, más adelante.

El procesamiento de palabras comienza llamando a la función `cleanWord()` que se encuentra en el módulo *Utilities*. Esta recibe un texto como parámetro y devuelve el texto ya procesado. Los pasos de los que consta este procesamiento son los siguientes:

1. Se eliminan todos aquellos símbolos que no sean caracteres alfabéticos, incluyendo las palabras que contienen números. Esto se hizo de esta manera porque había palabras que contenían números que, probablemente, fueron incluidos de forma accidental. Si estos datos los tuviéramos en cuenta, podría generar ruido en nuestro conjunto de entrenamiento al ser palabras no reconocidas por el diccionario anglosajón.

2. A continuación se hace una *tokenización*, que no es más que dividir el texto por espacios. El resultado que obtenemos es un vector con las palabras que componen el texto.

3. Aquellas palabras que tengan una longitud inferior o igual a dos se descartan. Consideramos que son palabras que no aportan nada al modelo al ser, en su mayoría, preposiciones o conjunciones. Para conseguir un conjunto uniforme, todas las palabras se cambian a minúsculas.

4. A continuación se lematizan las palabras. ¿En qué consiste la lematización? Lo explicaremos a continuación.

El término lematización proviene de lema. Un lema se entiende como cualquier entrada que se encuentre en el diccionario, para definir a una palabra dada. Por ejemplo, si realizamos una búsqueda en un diccionario online de un verbo conjugado, siempre nos aparecerá el infinitivo del verbo. ¿Qué es lo que ha sucedido?, pues que ese verbo se ha lematizado, obteniendo, por tanto, su lema. Pongamos un ejemplo en inglés:

"*He is walking to school with his friends*"

Si aplicamos la lematización a todas las palabras de esta frase, obtendríamos una oración similar a la que aparece en (5.1):

"*He be walk to school with his friend*" (5.1)

Como vemos, el verbo ha pasado de estar conjugado a estar en infinitivo. Aunque este, no ha sido el único cambio. También se ha cambiado *friends* por *friend*. Como podemos observar, un sustantivo que estaba en plural y un verbo que estaba conjugado han pasado a estar tal y como los podríamos encontrar en una entrada de diccionario.

El poder hacer este tipo de transformaciones resulta extremadamente útil, porque permite que los algoritmos de clasificación no hagan distinciones entre palabras que en realidad aportan el mismo conocimiento. Como resultado de esto, se conseguirá que los correos compartan más léxico, lo cual, puede ayudarnos en la tarea de clasificación.

5. Por último, se eliminan todas aquellas palabras que sean consideradas *stopwords*. Las *stopwords* es el término que se suele utilizar para denominar a aquellas palabras que no aportan ningún significado. Por ejemplo la palabra: *and, the* o *one* son casos de palabras que no aportan ningún valor al modelo porque se usan independientemente de la categoría que se pretenda clasificar.

Como en nuestro caso tendremos tres categorías, este proceso se repetirá con cada uno de los correos que haya en cada categoría y, evidentemente, con el conjunto de entrenamiento y el de validación.

Una vez termina el procesamiento de datos, estos son exportados en formato JSON. Esto se hará una única vez, pues extraer los datos del módulo *dataset* de *Sklearn* y preprocesarlos cada vez que ejecutáramos nuestro algoritmo genético no era eficiente. Por ello, se optó por generar una rutina que descargara los datos una sola vez, los procesara, y posteriormente los guardara este formato.

El formato para la salida y entrada fue elegido por ser, prácticamente, un estándar y por su similitud con los diccionarios de *Python* [17] lo cual, facilitará la importación cuando se vayan a utilizar en el algoritmo genético. Cabe destacar que, si por alguna razón, cuando se inicia la aplicación se detecta que no hay ningún fichero de entrenamiento o de validación, se ejecutará esa rutina de manera automática.

Como hay diferentes categorías, en el fichero almacenaremos una clave para cada una de ellas. El valor de cada clave será un vector que contendrá el conjunto de correos electrónicos pertenecientes a la categoría en cuestión. Cada uno de estos correos electrónicos se representará con los siguientes metadatos:

- ***category***: indica la categoría a la que pertenece el correo.
- ***from_***: indica la dirección de la que proviene el correo.
- ***subject***: corresponde al campo asunto del correo.
- ***organization***: indica la organización de la que proviene.
- ***msg***: es el contenido del correo, la mayor cantidad de palabras se encuentra en este campo.

Cuando importamos el conjunto de datos, creamos un objeto `DataCategory` por cada categoría presente en el conjunto. De tal forma que para el conjunto de entrenamiento tendremos tres objetos `DataCategory` y para el conjunto de validación otros tres más.

Es en este momento cuando podremos seleccionar los campos que más nos interesan. Esto se realiza de una manera muy sencilla. Tan solo se debe especificar un vector con los metadatos que queremos tener en cuenta. Por ejemplo:

```
keys_to_use: list = ['from_', 'subject', 'msg']
```

Una vez decididos los metadatos que queremos, el objeto `DataCategory` guardará la siguiente información:

- **Nombre de la categoría**: corresponde a la categoría a la que pertenecen los correos electrónicos que se almacenen.
- **Conjunto de correos electrónicos**: almacena los metadatos que se seleccionen de los correos, de tal forma que se conformará un string en el que se encuentren concatenados cada uno de estos metadatos. Ver Figura 5.3.
- **Corpus de palabras**: representa el conjunto de palabras presente en el conjunto anterior. Solo se almacenan palabras únicas de cada categoría.

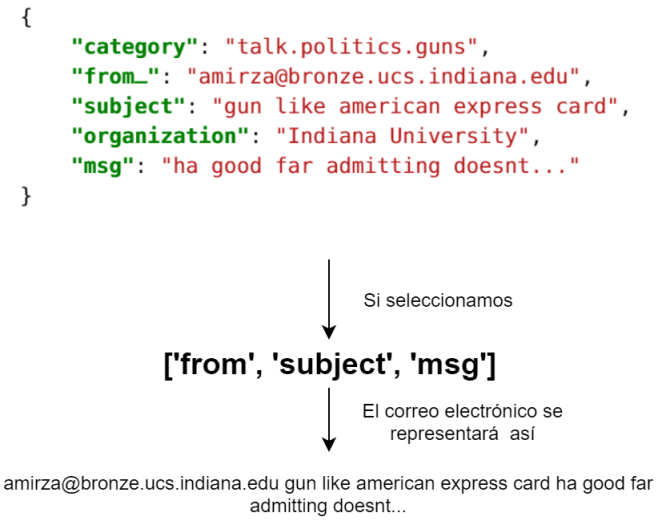


Figura 5.3: Representación de la concatenación de metadatos.

## 5.2. Módulo *Genetic*

En la presente sección se explicará el funcionamiento del algoritmo genético y como está integrado en el resto de la aplicación. Además, se profundizará en la implementación del mismo.

Antes de empezar con la instancia y ejecución del algoritmo genético, se genera la configuración con la que se lanzará. La clase `GeneticAlgorithmSpecification` será la encargada de almacenar esta configuración. Según el objetivo de los parámetros que se especifican cuando generamos la instancia, podemos dividirlos en dos grupos: parámetros para la **configuración de la implementación** y parámetros para la **configuración de la ejecución**.

En primer lugar hablaremos de los parámetros relacionados con **configuración de la implementación**. Esta configuración permite intercambiar y combinar las clases que se usarán para representar a cada uno de los componentes del algoritmo genético. Por ejemplo, podemos elegir el método para seleccionar los padres o elegir como será el método de supervivencia tan sólo cambiando una línea de código. Además, cualquier clase que herede de un tipo básico quedará registrado en una clase que se encarga de ello, pudiendo saber todos los tipos de clases existentes para representar cada componente.

Esto dota al programa de una gran flexibilidad y, a su vez, permite desarrollar nuevas clases para cada uno de los componentes del algoritmo genético sin tener que cambiar en exceso, el funcionamiento general. Las variables o parámetros que se pueden configurar son los siguientes: **tipo de cromosoma, tipo de gen, tipo de individuo, operador de reemplazo, operador de selección de padres, función de penalización, operador de cruce, operador de mutación y función generadora de la población**.

Con el objetivo de que el código se pueda mantener y extender, todas las clases que representan a un mismo componente (de una forma o de otra), usan métodos comunes. Pongamos un ejemplo con el operador de selección de padres:

> La selección de padres cuenta con una clase "base"que simula una interfaz (tal y como pasa con otros lenguajes de programación como *Java*). Esta "interfaz"es compartida por todas las clases que heredan de esta clase "base". Imaginemos que queremos definir una nueva clase que seleccione un individuo de la población con el método *roulette wheel*. Tan sólo tendríamos que definir una nueva clase, heredar de la clase "basez dar una definición para el método de selección de padres.
>
> Para usar esta esta nueva clase, tan solo bastaría con especificarlo cuando se establece la configuración para el algoritmo genético (Ver Apéndice A.6), y el programa se encargará de buscar la definición y ejecutar el código correspondiente.

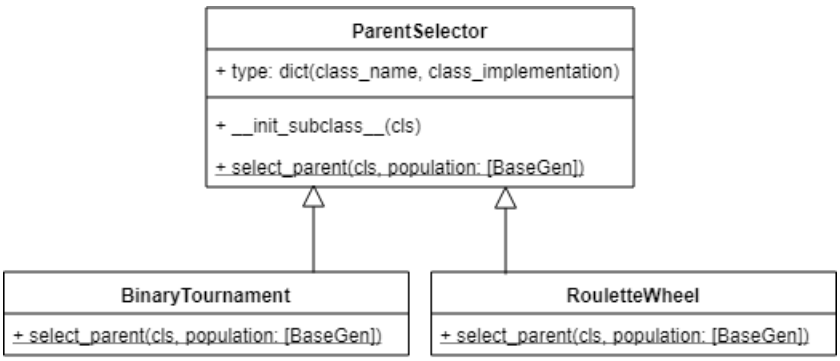


Figura 5.4: Diagrama UML de las clases para la selección de padres.

En cuanto a la **configuración del funcionamiento** del algoritmo, cabe destacar que se pueden personalizar las siguientes variables:

- **Probabilidad de mutación**: representa la probabilidad de que un gen se vea alterado. Es un valor entre 0 y 1.
- **Probabilidad de cruce**: se corresponde a la probabilidad de que se produzca un cruce entre dos individuos. Es un valor entre 0 y 1.
- **Tasa de penalización**: los individuos que no cumplan con ciertas restricciones verán su *fitness* penalizado gracias a la función de penalización. La tasa de penalización está relacionada con lo estricto que se desee ser con la restricción que se imponga. Es un valor entre 0 y 1.
- **Porcentaje de palabras**: es un número entre 0 y 1. Su función es determinar la cantidad de palabras que se escogerán de cada categoría para representar a una posible solución.
- **Tamaño de la población**: es un número entero. Indica la cantidad de individuos o soluciones con las que se trabajará.
- **Número de generaciones**: es un número entero. Indica el número de iteraciones del algoritmo genético. Una vez alcanzado, se detiene su ejecución y se devuelve el mejor individuo de la población.

A continuación se explicará la implementación de cada uno de los componentes del algoritmo.

### 5.2.1. Individuo

Un individuo representa una solución. En nuestro problema, este individuo se encuentra compuesto por un valor *fitness* y un cromosoma. Este valor determinará lo buena que es una solución. Por otro lado, el cromosoma contendrá las palabras que representarán a cada una de las categorías.

Nuestra clase base que representa a un individuo se denomina `BaseIndividual` y almacena esta puntuación en notación decimal (siempre entre 0 y 1) y el cromosoma del individuo, que será de tipo *BaseChromosome*. Esta última clase contiene la siguiente información:

- **Vector de genes seleccionados**: es un vector de objetos de tipo *BaseGen*. Los objetos de este tipo almacenan una palabra y la categoría a la que pertenece la palabra. Haciendo analogía con un cromosoma binario, estos genes representarían los 1's.
- **Vector de genes eliminados**: es otro vector del mismo tipo que el anterior. Representa todas aquellas palabras que no están seleccionadas pero, que, en algún momento de la ejecución, podrían estarlo. Haciendo analogía con cromosomas de tipo binario, estos genes representarían los 0's.

Por tanto, nuestro cromosoma, se encuentra representado por el primero de los vectores. Destacar que los genes seleccionados no se encuentran contenidos en el vector de genes eliminados ni viceversa.

Pero, ¿por qué se ha optado por representar el cromosoma con dos vectores cuando en realidad se podría hacer con uno? La respuesta está en que, teniendo estos dos vectores, los cálculos que realiza el procesador para determinar cuáles son los genes seleccionados y cuáles los eliminados se reducen significativamente. Sin embargo, el consumo de memoria aumenta al tener que almacenar más información, pero esto es preferible antes que gastar recursos del procesador. Esto se pensó de esta manera porque, posteriormente, los genes seleccionados se consultarán habitualmente, y hacerlo así, hace que tan sólo se devuelva un vector, sin tener que buscar cuáles son los genes seleccionados. Por supuesto, cuando se elimina o se añade un gen, se mantiene la coherencia del modelo, añadiendo y eliminando de un vector o de otro según corresponda.

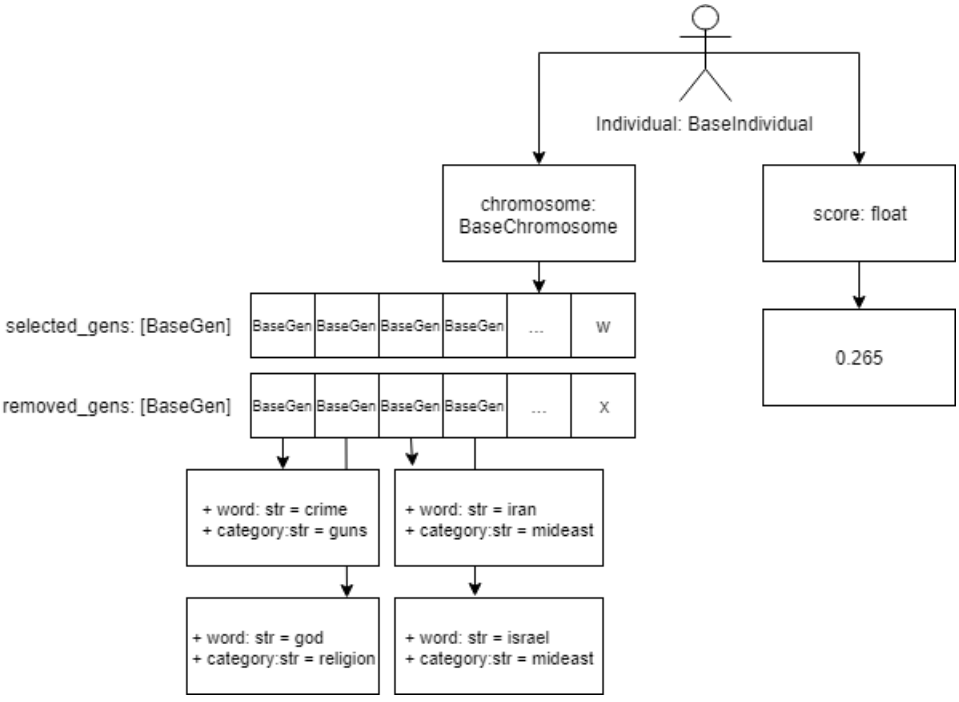


Figura 5.5: Esquema de la clase individuo.

A continuación explicaremos como se generarán los individuos de la población inicial y cómo se alimentará la clase `BaseChromosome`.

### 5.2.2. Generación de la población inicial

Como decíamos anteriormente, uno de los parámetros es el porcentaje de palabras que se usará para representar a cada una de las categorías. La representación de una categoría viene determinada por el conjunto de genes seleccionados que pertenecen a esa misma categoría. Ver Figura 5.5.

Lo primero que haremos es generar un diccionario *Python* de $genesPosibles$ en el que cada clave se corresponderá con una categoría, y su valor, será el conjunto de `BaseGen` posibles para esa categoría. Como se aprecia en la Figura 5.5, cada uno de los `BaseGen` contendrá una palabra y la categoría a la que pertenece. Al final de este proceso tendremos un diccionario con tantas entradas como categorías haya. El tamaño del vector de genes posibles coincidirá con el número de palabras que tenga el corpus de la categoría en cuestión.

Como cada categoría puede tener un número diferente de palabras, se trabaja con porcentajes. Este porcentaje $y$ indicará la cantidad de genes que deberían ser seleccionados de cada una de las categorías para conformar el cromosoma o vector de genes seleccionados de un individuo.

Una vez inicializado el diccionario $genesPosibles$, se calcula la cantidad de genes que deben representar a cada categoría, teniendo en cuenta el porcentaje $y$. Una vez calculado, guardamos la distribución en un nuevo diccionario al que llamaremos $distribuciónIdeal$.

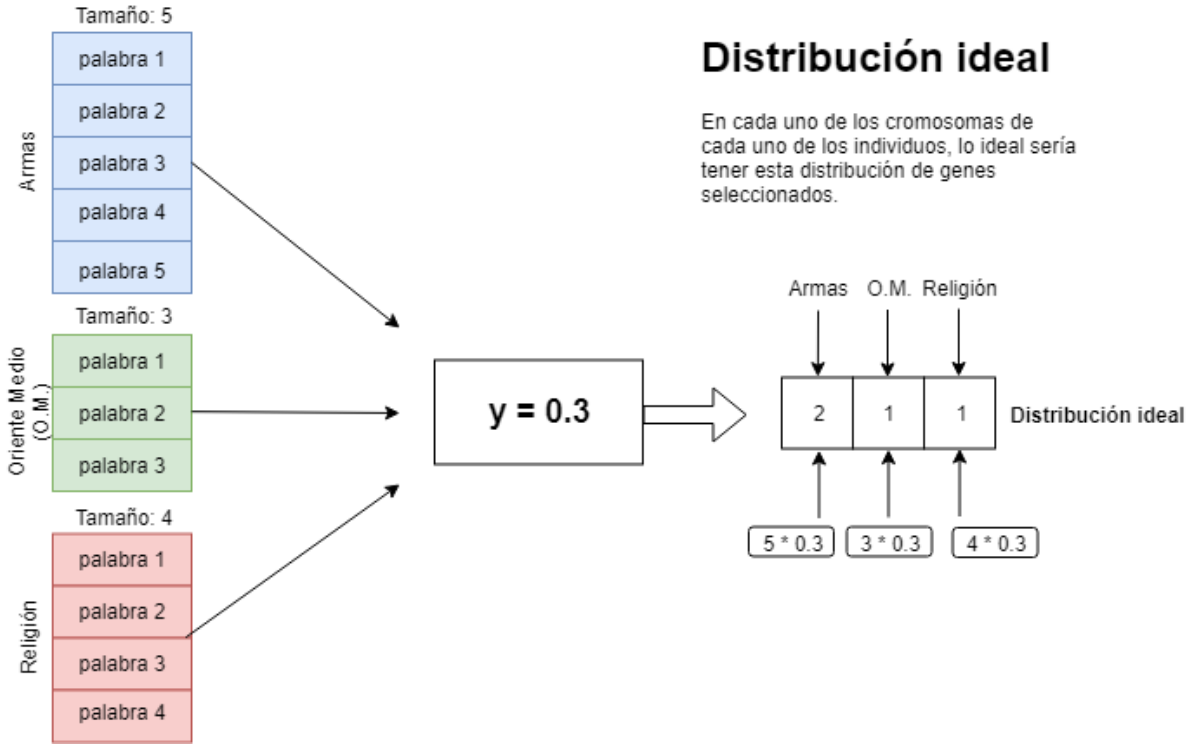


Figura 5.6: Esquema para la generación de la distribución ideal.

Cuando ya tenemos el diccionario $distribuciónIdeal$ podemos optar por restringir que todos los individuos que generemos sigan estrictamente esta distribución, o dar cierto margen de libertad y permitir que esta restricción sea débil, consiguiendo así que hayan individuos cuyos genes seleccionados no sigan estrictamente la distribución ideal. Según

optemos por una opción u otra podremos producir la población según una distribución aproximada o exacta.

- **Distribución aproximada**. No se cumple estrictamente la distribución ideal de genes seleccionados. En su lugar, se selecciona un número de genes próximo a la distribución ideal con una función aleatoria. Por ejemplo, la probabilidad de que un gen de la categoría $c_1$ sea escogido vendrá determinado una expresión que relaciona el número de genes que contiene la categoría $|genesPosibles_{c1}|$ con la distribución ideal de la misma $distribuciónIdeal_{c1}$:

$$umbral_{c_1} = \frac{distribuciónIdeal_{c1}}{|genesPosibles_{c1}|} \tag{5.2}$$

  A continuación se recorre cada uno de los genes posibles del vector $genesPosibles_{c1}$ y, en cada iteración $i$, se genera un número aleatorio $a$ entre 0 y 1.

$$\begin{cases} a < umbral_{c_1} & \text{Introducimos el gen } i \text{ en una lista de genes seleccionados} \\ a >= umbral_{c_1} & \text{Introducimos el gen } i \text{ en una lista de genes posibles} \end{cases}$$

  Una vez recorrido el vector $genesPosibles_{c1}$ se repite el proceso con cada una de las categorías, teniendo al final un único vector de genes seleccionados que conformará nuestro cromosoma y otro vector de genes posibles en los que habrán genes de todas las categorías. Con estos dos vectores construiremos nuestro `BaseChromosome` y, finalmente, nuestro individuo.

  Debido al uso de una función aleatoria y al umbral, el número de genes seleccionados en los individuos de la población será, probablemente, diferente.

- **Distribución exacta**. Se cumple de manera estricta la distribución ideal de genes seleccionados. Para cada categoría se seleccionan tantos genes como indica la distribución ideal. En este caso todos los individuos tendrán cromosomas con la misma distribución de genes seleccionados. Para este caso los genes seleccionados se escogen de manera aleatoria hasta alcanzar la cantidad que indique la $distribuciónIdeal$ para la categoría en cuestión.

Tanto si se elige la distribución aproximada, o la exacta, el proceso se repetirá tantas veces como individuos queramos que haya en nuestra población.

Como producto de las sucesivas operaciones de cruce, la distribución de genes seleccionados para cada categoría se verá alterada en la mayoría de las ocasiones, aunque esto dependerá de la configuración de implementación que elijamos. Esto se explicará mas en detalle cuando se explique el operador de cruce y el de mutación, pero ya adelantamos que habrá una función de penalización que afectará a la puntuación del individuo si este no cumple una serie de restricciones.

### 5.2.3. Función de evaluación o *fitness*

Una vez generada la población, se evalúa cada uno de los individuos para conocer la calidad de la solución. Para ello, hacemos uso de dos clasificadores disponibles en la librería *Sklearn* [14]. Para compactar más el código, se hace uso de un "*Pipeline*".

```
text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', OneVsRestClassifier(LinearSVC(class_weight="balanced")))
])
```

Los dos clasificadores usados son *LinearSVC* [18] y *OneVsRestClassifier* [19], y para alimentarlos, se necesitan dos conjuntos de datos. Un conjunto de datos de entrenamiento y otro de validación. El conjunto de datos de entrenamiento serán los genes seleccionados del individuo. Recordemos que cada uno de estos genes contiene una palabra y la categoría a la que pertenece dicha palabra.

Por otro lado, también necesitamos un conjunto de validación. En este caso, el conjunto de validación será el conjunto de correos electrónicos o documentos almacenados en cada uno de los objetos de tipo `DataCategory` de los que hablamos al principio de esta sección. De esta manera sabremos cómo se comporta nuestra solución al clasificar un conjunto de correos.

La librería nos proporciona diferentes métricas para evaluar la calidad del conjunto de entrenamiento. Nosotros hemos evaluado la calidad de la solución en base al valor-F, pero se podrían haber usado otro tipo de métricas para este fin. El valor-F se define tal y como figura en la Ecuación (5.3).

$$valor\text{-}F = 2 * \left( \frac{precision * recall}{precision + recall} \right) \tag{5.3}$$

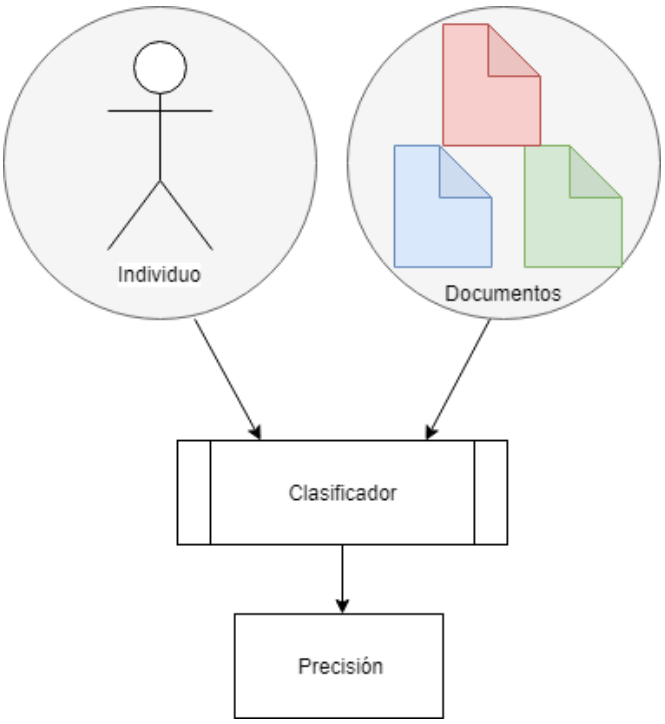


Figura 5.7: Esquema funcionamiento clasificador.

### 5.2.4. Función de penalización

La utilidad de la función de penalización es reducir el *fitness* de aquellos individuos que no cumplan con una serie de condiciones. En nuestro caso perseguimos dos objetivos de manera simultánea.

1. Acotar la cantidad de palabras que representan a una categoría.
2. Intentar mantener la distribución de palabras seleccionadas lo más similar posible a la distribución ideal.

Ambas restricciones son débiles, pues en realidad, tampoco importa demasiado que el conjunto de genes seleccionados varíe su tamaño ligeramente respecto a la distribución ideal, pues nuestro problema se enfoca en encontrar un conjunto de palabras reducido que pueda representar bien a cada una de las categorías y que nos permita obtener buenos resultados clasificando un conjunto de validación.

Lo mismo sucede con la distribución ideal de palabras seleccionadas. En un principio la intuición nos dice que lo ideal es que, si cada categoría tiene un número de palabras diferente, se represente proporcionalmente a cada una de las categorías. Pero haciendo esto, suponemos que la calidad del conjunto de palabras que hay para cada categoría es igual y esto puede ser que no sea así. De hecho, después del procesamiento de datos se dan circunstancias en las que aparecen palabras que no significan nada. Esto se debe a que el procesamiento no es capaz de detectar todas las palabras "inútiles", por ello que resulta vital que se dé una cierta libertad para que se altere la distribución de palabras que representa a cada categoría.

Definamos entonces, cómo funciona la penalización y para qué sirve la tasa de penalización. Para ello, necesitaremos tener a un individuo, y por ende, un cromosoma. Además, tendremos que saber cómo es nuestra distribución ideal.

Imaginemos que tenemos un conjunto $C$ de categorías compuesto por 3 categorías, que nuestra distribución ideal de palabras es $di_{c1} = x, di_{c2} = y, di_{c3} = z$ y que estamos evaluando un individuo con la siguiente distribución $d_{c1} = x + w, d_{c2} = y + b, d_{c3} = z$. Es evidente que la distribución de categorías no es la ideal, entonces tendremos que aplicar una penalización que se calculará tal y como se puede apreciar en el Algoritmo 2.

**Algoritmo 2** Pseudocódigo para la función de penalización

```
1: function penalizar(individuo, distribución_ideal)
2:     Penalización_total ← 0
3:     for i en C do
4:         Penalización_total ← Penalización_total + |di_ci − d_ci|
5:     individuo_puntuación ← individuo_puntuación − Penalización_total ∗ tasa_penalización
```

Como podemos observar, la utilidad de la tasa de penalización está en disminuir o aumentar la tasa de penalización. Si la $tasa_penalización = 0{,}5$, entonces $individuo_{puntuación}$ se multiplicará por ese factor, y la restricción se hará más débil (pues la penalización se dividirá entre dos).

### 5.2.5. Selección de padres

Se han implementado diferentes clases para la selección de padres, sin embargo, nos centraremos en la selección por torneo binario [20]. La selección de padres se realiza seleccionando dos individuos aleatorios de la población y escogiendo al mejor de ellos según su puntuación. Este proceso se repite de nuevo para escoger al segundo padre, puesto que nuestro algoritmo genético selecciona siempre a dos padres.

### 5.2.6. Operador de cruce

El cruce es la operación por la cual dos individuos intercambian información genética y como resultado de este intercambio se genera un individuo con diferente información genética. Para nuestro problema se ha implementado el cruce uniforme.

En primer lugar, el hijo resultante del cruce tendrá un cromosoma (o conjunto de genes seleccionados) del mismo tamaño que el cromosoma más pequeño de los padres. Es decir, si tenemos un padre $p_1$ con una cantidad de genes seleccionados $|p_{1s}|$ y otro padre $p_2$ con $|p_{2s}|$ genes seleccionados, la cantidad de genes que se seleccionarán para conformar el cromosoma del hijo vendrá determinado por $min(|p_{1s}|, |p_{2s}|)$.

El cromosoma del nuevo individuo se generará de la siguiente manera:

1. Se concatenan los dos vectores de genes seleccionados de ambos padres y se genera uno nuevo.

2. A continuación, se emplea una función que permite desordenar aleatoriamente este vector.

3. Una vez que está desordenado, se crea un nuevo vector que contendrá los genes del descendiente o hijo.

4. El vector anterior se rellenará eligiendo aleatoriamente índices del vector que contiene todos los genes de ambos padres.

Cabe destacar que, en el proceso de cruce, se permite que el individuo tenga una distribución de genes diferente a la de los padres y a la ideal.

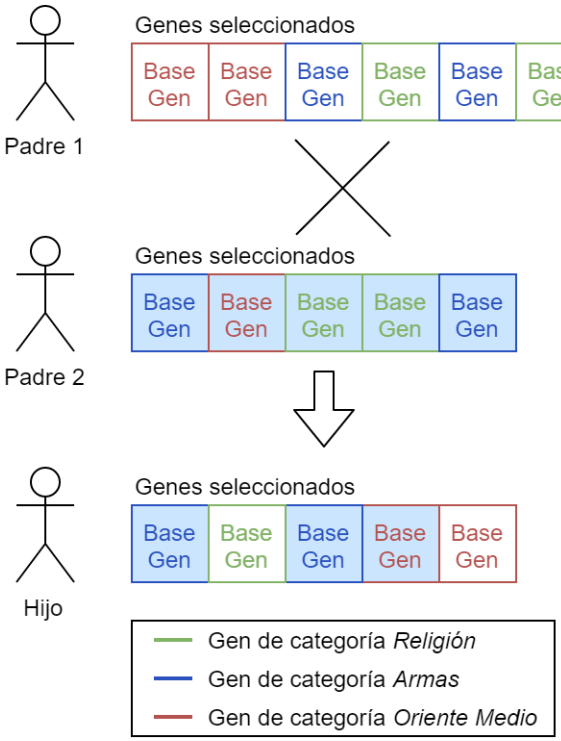


Figura 5.8: Esquema de funcionamiento del cruce.

### 5.2.7. Operador de mutación

La mutación es la operación por la cual se selecciona uno o varios genes de la solución y se alteran. Esto podría ocasionar que, eventualmente, se mejore la puntuación de una solución. En nuestro caso esta mutación tratará de eliminar o añadir genes a nuestra solución.

Cuando añadimos un gen, este se borra del vector de genes eliminados y pasa al de genes seleccionados. Si por el contrario, eliminamos un gen, este se borra del vector de genes seleccionados y pasa al de eliminados. Ver Algoritmo 3.

**Algoritmo 3** Pseudocódigo para la mutación

```
1: function mutar(individuo, tasa_mutación)
2:     Cromosoma ← get(Individuo, Cromosoma)
3:     Genes_Seleccionados ← getSize(Cromosoma, Genes_Seleccionados)
4:     Genes_Eliminados ← getSize(Cromosoma, Genes_Eliminados)
5:
6:     Prob_Eliminar ← tasa_mutación
7:     Prob_Insertar ← tasa_mutación * Genes_Seleccionados/Genes_Eliminados
8:
9:     for gen en Cromosoma do
10:        Umbral ← numero_aleatorio(0, 1)
11:        if Umbral < Prob_Eliminar ∧ seleccionado(Cromosoma, gen) then
12:            eliminar(Cromosoma, gen)
13:        else if Umbral < Prob_Seleccionar ∧ ¬seleccionado(Cromosoma, gen) then
14:            seleccionar(Cromosoma, gen)
```

Esta función de mutación, en la mayoría de ocasiones añade nuevos genes al vector de genes seleccionados, de ahí que hayamos usado la función de penalización previamente mencionada para contrarrestar este efecto.

### 5.2.8. Operador de reemplazo

Los supervivientes son aquellos individuos que pasarán a formar parte de la siguiente población. En nuestro caso tenemos una población inicial de tamaño $|p_i|$. Sobre esta población se repetirá el proceso de selección, cruce y mutación $\frac{|p_i|}{2}$ veces, pues siempre elegimos dos padres y producimos dos hijos. Todos los hijos resultantes del cruce de dos padres se almacenarán en una variable temporal, de tal forma que, cuando se acabe de recorrer toda la población $p_i$, tendremos una población hija $p_h \mid |p_i| = |p_h|$.

Una vez tenemos las dos poblaciones, las concatenamos y ordenamos en orden descendente según la puntuación de cada uno de los individuos. Finalmente escogemos los $|p_i|$ primero individuos, teniendo así, los mejores individuos de ambas poblaciones.

En este caso, estaríamos seleccionando los individuos de la siguiente población de forma elitista. Sin embargo, también se implementó una estrategia generacional híbrida, en la que buscamos el mejor de los individuos $p_i + p_h$ y eliminamos el peor individuo de $|p_h|$ para insertar el mejor de ambas poblaciones. Por tanto, conservaríamos los $|p_h| - 1$ hijos y el mejor individuo que haya en ambas poblaciones.

## 5.3. Herramientas utilizadas

Las herramientas y tecnologías utilizadas son variadas y han ayudado a la realización de este TFG. A continuación, comentaremos cada una de ellas.

- ***Python*** [21]. Uno de los lenguajes más utilizados hoy en día para el tratamiento de datos y el desarrollo de aplicaciones relacionadas con la inteligencia artificial. Esto se debe a la extensa cantidad de librerías que permiten desarrollar código rápidamente y a la flexibilidad del propio lenguaje. Además, al soportar objetos,

permite organizar mejor la aplicación. Como todos los lenguajes interpretados, *Python* tiene el inconveniente de que no es tan rápido como los lenguajes compilados.

- ***Git & GitHub*** [22][23]. *Git* es un software de control de versiones que resulta imprescindible para llevar un control de cambios en el proyecto. Por su parte, *GitHub* permite alojar el proyecto en un repositorio en la nube con el fin de que sea fácilmente accesible. Además, resulta de utilidad para visualizar los cambios y la estructura general del proyecto.

- ***GitHub Desktop*** [24]. Sin duda, una herramienta muy útil para llevar un control de cambios a través de una interfaz sencilla e intuitiva, en lugar de trabajar a través de la consola. Como curiosidad, mencionar que está programado en *Electron*, un framework desarrollado por el equipo de *GitHub* que permite realizar aplicaciones gráficas haciendo uso de *Node.js* y *Chromium* del lado del servidor [25].

- ***PyCharm Community IDE*** [26]. Es un *IDE* para la programación de aplicaciones en *Python* desarrollado por *JetBrains*. En los inicios del proyecto se comenzó usando *Eclipse*, pues a través de plugins puede soportar *Python*. Sin embargo este soporte no era del todo bueno y el autocompletado no ayudaba demasiado. Por ello, se comenzó a usar este *IDE*, muy recomendado para desarrollar aplicaciones en *Python*.

- ***Visual Studio Code*** [27]. Editor de código por excelencia en numerosos proyectos. Está desarrollado por *Microsoft* y cuenta con una amplísima extensión de *plug-ins* que hace que sea muy flexible y polivalente. En el proyecto se usó para visualizar los diferentes ficheros *JSON* de entrada y salida.

- ***Scikit-learn*** [14]. Es una librería para *Python*. Cuenta con un sinfín de métodos útiles para aplicar *machine learning*. No se ha usado demasiado en la aplicación, pero, sin duda, juega un papel crucial por ser el mecanismo con el que evaluamos la calidad de una solución.

- ***NLTK*** [28]. Se trata de otra librería para *Python* usada para el procesamiento de lenguaje natural. Esta librería también se usó puntualmente, pero sin duda ha ayudado a agilizar ciertas tareas que de no ser así, se tendrían que haber implementado de cero.

- ***Kaggle*** [29]. Es de las páginas sitios más conocidos para la obtención de conjunto de datos. Hay una gran comunidad detrás y además se lanzan retos relacionados con el tratamiento de datos, para competir entre los miembros de la comunidad. Tal es así, que muchas grandes empresas lanzan retos en los que se recompensa a las personas o el equipo de personas que consiga obtener unos mejores resultados. Actualmente es propiedad de *Alphabet*, la matriz de *Google*.

- ***Matplotlib*** [30]. Se trata de una libería para *Python* que permite visualizar de manera sencilla y rápida un gráficos. Se usó para elaborar las gráficas que se verán en el siguiente capítulo. Además, durante la ejecución del algoritmo genético, se da la opción de mostrar una interfaz gráfica que fue elaborada con esta librería, y con la que se puede visualizar la evolución de la población.

# Capítulo 6

# Análisis de resultados

En este Capítulo se describirán las pruebas realizadas, así como el conjunto de datos utilizado para el entrenamiento y la validación. Además, se detallará cada una de las configuraciones con las que se probó el algoritmo y el entorno de pruebas utilizado para la ejecución de las mismas.

Python [21] no es el mejor lenguaje para hacer tareas donde el tiempo de cómputo sea vital. En este sentido, los lenguajes compilados y de más bajo nivel como C o C++, pueden ser más interesantes cuando hacemos este tipo de tareas pesadas. Sin embargo, en nuestra aplicación, el tiempo de cómputo no es crucial, pues tratamos de buscar un conjunto de palabras reducido que pueda determinar, con unos buenos resultados, la categoría de correos entrantes. Pero, lo que sí podría resultar crucial de cara a implementar una solución comercial, sería la rapidez con la que un correo o conjunto de correos sea clasificado una vez que ya se tiene el modelo o conjunto de palabras más representativas de cada una de las categorías.

Todas las pruebas han sido ejecutadas bajo el siguiente entorno:

- **Procesador:** Intel Core i7 6700k de 4 núcleos y 8 hilos a una frecuencia de 4.00GHz.
- **Memoria RAM:** 8 GB de memoria RAM DDR4 a 2133 MHz.
- **Versión del intérprete de Python:** Python 3.7.1 para 64 bits.
- **Versión de la librería Sklearn:** Sklearn 0.20.3
- **Versión de la librería nltk:** nltk 3.4

En cuanto al conjunto de entrenamiento, encontramos las tres categorías que previamente mencionamos. En total suman 36.595 palabras y se reparten de la siguiente manera:

- **Religión (soc.religion.christian)**: contiene 11.734 palabras únicas después del proceso de limpieza.
- **Armas (talk.politics.guns)**: contiene 10.953 palabras únicas después del proceso de limpieza.
- **Oriente Medio (talk.politics.mideast)**: contiene 13.908 palabras únicas después del proceso de limpieza.

Según el objetivo que tienen las pruebas, pueden ser divididas en dos grupos: Pruebas de clasificación y Pruebas de evolución. El primer grupo de pruebas se realizaron para evaluar el algoritmo genético, mientras el segundo grupo, evalúa la mejor solución aportada por el algoritmo genético con un conjunto de validación.

## 6.1. Pruebas de evolución

Como decíamos anteriormente, este conjunto de pruebas se realizó con el fin de evaluar la evolución de las soluciones del algoritmo genético. A continuación detallaremos los diferentes parámetros con los que se probó el algoritmo:

- **Tasa de cruce (CR):** 80 % y 100 %.
- **Tasa de mutación (MR):** 5 % y 10 %.
- **Porcentaje de genes seleccionados (ISL):** 30 % y 20 %.
- **Tamaño de la población (PL):** 30 y 50 individuos.
- **Número de generaciones:** se fijó un valor de 150 generaciones, ya que previamente se habían hecho pruebas para comprobar en cuántas generaciones se producía la convergencia y se determinó que, en la mayor parte de los casos, a partir de 150 generaciones ya no había mejoras significativas.

La configuración de clases que se usó para representar a cada componente del algoritmo genético fue la siguiente:

- **Tipo de cromosoma**: se usó la clase `BaseChromosome`.
- **Tipo de gen**: se usó la clase `BaseGen`.
- **Operador de reemplazo**: elitista. Se corresponde con la clase `BestIndividuals`.
- **Selección de padres**: por torneo binario. Se corresponde con la clase `Tournament`.
- **Función de penalización**: se penaliza que la distribución no sea la ideal. Se corresponde con la clase `PenaltyDistribution`.
- **Tipo de cruce**: cruce uniforme. Se corresponde con la clase `BaseChromosome`.
- **Tipo de mutación**: mutación controlada, descrita en la Sección 5.2.7. Se corresponde con la clase `ControlledMutation`.
- **Generador de población**: individuos con cromosomas de distribución aproximada. Se corresponde con la clase `ApproximatedSize`.

Las pruebas se han realizado haciendo todas las combinaciones posibles de parámetros anteriores, por tanto se probaron $2^4 = 16$ configuraciones diferentes. Como los algoritmos genéticos son estocásticos se han realizado 10 ejecuciones por cada una de estas configuraciones, ejecutando el algoritmo un total de $16 * 10 = 160$ veces. A continuación, en la Tabla 6.1 se muestra el valor mínimo, máximo, media, mediana y desviación típica del *fitness* del mejor individuo para cada una de las 16 configuraciones. La tabla se encuentra

ordenada según el valor medio de *fitness* obtenido para las 10 ejecuciones de cada una de las configuraciones.

| Configuración | Mínimo | Máximo | Media | Mediana | Desviación |
|---|---|---|---|---|---|
| CR: 0.8, MR: 0.05, ISL: 0.3, PL: 50 | 0.8538 | 0.8959 | 0.8734 | 0.8751 | 0.011 |
| CR: 0.8, MR: 0.05, ISL: 0.3, PL: 30 | 0.8538 | 0.8864 | 0.8697 | 0.8689 | 0.0096 |
| CR: 0.8, MR: 0.05, ISL: 0.2, PL: 50 | 0.8538 | 0.8763 | 0.8652 | 0.8636 | 0.0075 |
| CR: 0.8, MR: 0.1, ISL: 0.2, PL: 30 | 0.7929 | 0.8959 | 0.8603 | 0.8688 | 0.0283 |
| CR: 0.8, MR: 0.05, ISL: 0.2, PL: 30 | 0.8538 | 0.8732 | 0.8602 | 0.8587 | 0.0051 |
| CR: 1.0, MR: 0.05, ISL: 0.3, PL: 50 | 0.7929 | 0.8959 | 0.858 | 0.8605 | 0.0254 |
| CR: 1.0, MR: 0.05, ISL: 0.3, PL: 30 | 0.7929 | 0.8959 | 0.8556 | 0.8587 | 0.0251 |
| CR: 1.0, MR: 0.1, ISL: 0.2, PL: 30 | 0.7888 | 0.8959 | 0.8537 | 0.8587 | 0.0286 |
| CR: 0.8, MR: 0.1, ISL: 0.2, PL: 50 | 0.7929 | 0.8959 | 0.8529 | 0.8636 | 0.0309 |
| CR: 1.0, MR: 0.05, ISL: 0.2, PL: 50 | 0.7929 | 0.8959 | 0.8528 | 0.8576 | 0.0247 |
| CR: 1.0, MR: 0.05, ISL: 0.2, PL: 30 | 0.7929 | 0.8959 | 0.852 | 0.8571 | 0.0258 |
| CR: 0.8, MR: 0.1, ISL: 0.3, PL: 30 | 0.7929 | 0.8959 | 0.8518 | 0.8587 | 0.0289 |
| CR: 0.8, MR: 0.1, ISL: 0.3, PL: 50 | 0.7929 | 0.8959 | 0.8511 | 0.8565 | 0.0272 |
| CR: 1.0, MR: 0.1, ISL: 0.2, PL: 50 | 0.7888 | 0.8959 | 0.8508 | 0.8576 | 0.0295 |
| CR: 1.0, MR: 0.1, ISL: 0.3, PL: 30 | 0.7888 | 0.8959 | 0.8504 | 0.8566 | 0.0286 |
| CR: 1.0, MR: 0.1, ISL: 0.3, PL: 50 | 0.7888 | 0.8959 | 0.8504 | 0.8556 | 0.0277 |

Tabla 6.1: Análisis de los resultados por configuración.

Atendiendo a los resultados obtenidos en la Tabla 6.1, podemos observar que las tres primeras configuraciones son las mejores. Entre la segunda y tercera, la media del *fitness* alcanzado en las diez ejecuciones es muy similar. La configuración de la tasa de cruce, y mutación era la misma, lo que varió fue el porcentaje de palabras usadas para representar a cada categoría y la población con la que se trabajó. Por tanto, podemos afirmar que, líneas generales, la tasa de cruce y mutación que mejor ha funcionado es 80 % y 5 %, respectivamente.

En cuánto al porcentaje de palabras empleadas para representar a cada categoría, se puede apreciar que no resulta tan condicionante, como tampoco el número de individuos, pues en la tabla se aprecia un reparto más uniforme.

Por cuestiones de visualización no se pudo mostrar una gráfica con la evolución media de todas las configuraciones. Por esta razón, se escogieron las configuraciones según el resultado que se obtuvo al clasificar el conjunto de validación. Para que hubiera heterogeneidad, se eligió la primera, segunda, tercera, duodécima y decimoquinta configuración de la Tabla 6.3.

Las configuraciones anteriormente mencionadas son las representadas en la Figura 6.1.

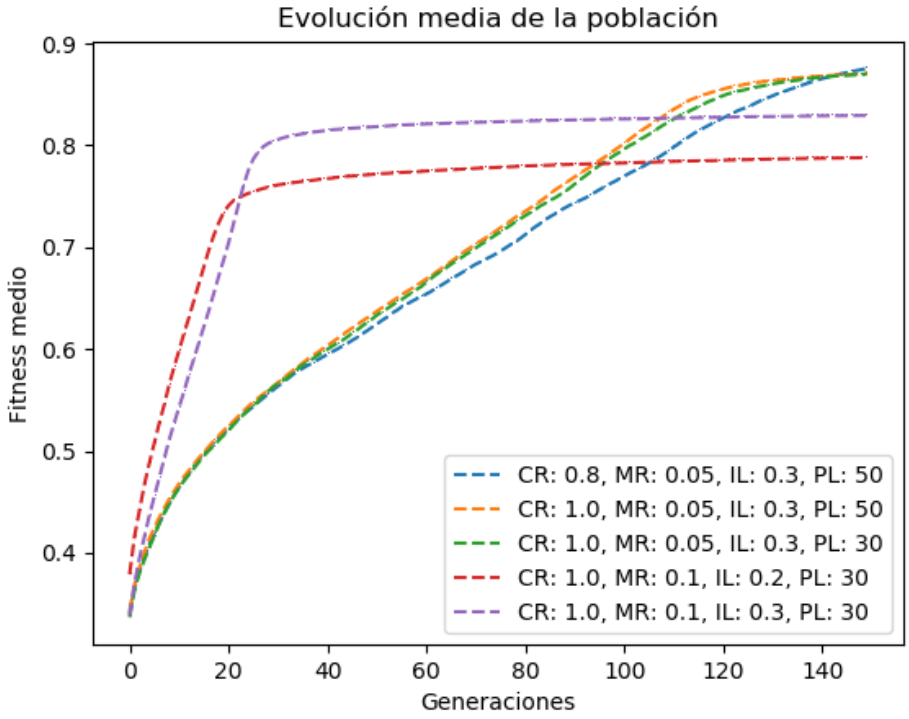


Figura 6.1: Evolución de la puntuación media.

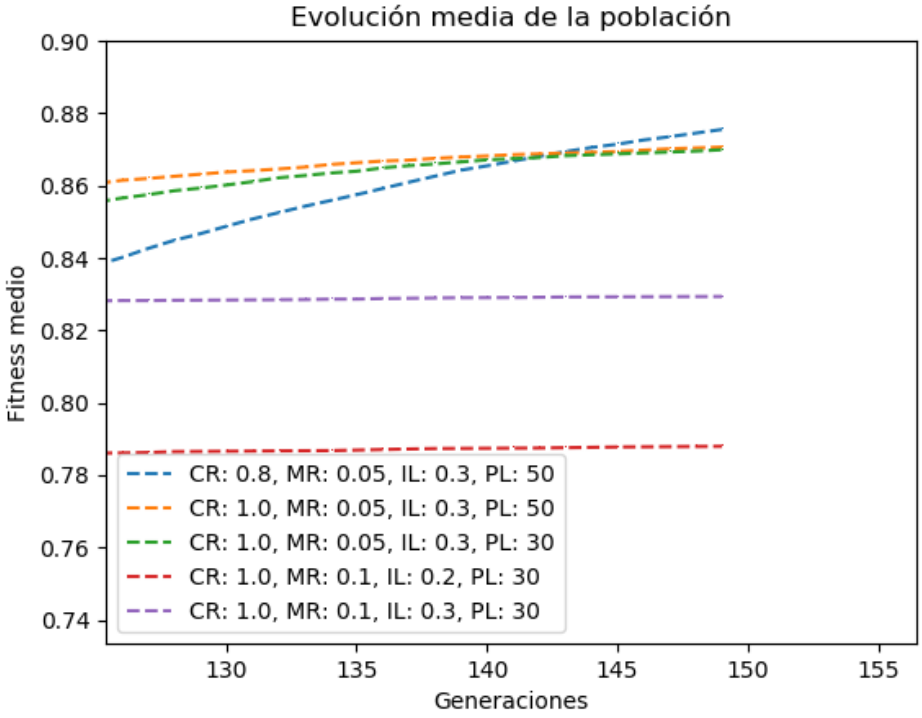


Figura 6.2: Detalle de la evolución del *fitness* media.

Como la población inicial la generamos con el método de distribución aproximada, la cantidad de genes seleccionados para cada una de las categorías suele ser inferior a la cantidad que hemos fijado como máximo (20 % o 30 % dependiendo de la configuración). A

continuación, en la Tabla 6.2 mostraremos el crecimiento de las palabras que representan a las diferentes categorías. En todos los casos se puede apreciar como la función de penalización acota el crecimiento del conjunto de palabras seleccionadas, quedándose todas ellas en sus respectivos máximos 20 % o 30 %.

| **Configuracion** | **Palabras iniciales** | **Palabras finales** | **% palabras inicial** | **% palabras final** |
|---|---|---|---|---|
| CR: 0.8, MR: 0.05, ISL: 0.3, PL: 50 | 8447 | 10955 | 23 % | 30 % |
| CR: 1.0, MR: 0.05, ISL: 0.3, PL: 50 | 8444 | 11010 | 23 % | 30 % |
| CR: 1.0, MR: 0.05, ISL: 0.3, PL: 30 | 8440 | 11010 | 23 % | 30 % |
| CR: 1.0, MR: 0.1, ISL: 0.2, PL: 30 | 6105 | 7463 | 16 % | 20 % |
| CR: 1.0, MR: 0.1, ISL: 0.3, PL: 30 | 8442 | 11164 | 23 % | 30 % |

Tabla 6.2: Crecimiento medio del número total de palabras que representan a las categorías.

Nos pareció interesante, ver cómo evolucionaba la cantidad de genes seleccionados y relacionarlo con la mejora de la función *fitness*. Como se puede apreciar, es evidente que hay una relación entre el número de palabras escogidas para representar a cada categoría y el *fitness*. Esto se debe a un factor totalmente lógico, y es que, cuantas más palabras se usen, mejor va a ser esta puntuación. Si nos fijamos en la Tabla 6.3 de la siguiente sección, podemos apreciar como la configuración *CR: 1.0, MR: 0.1, ISL: 0.3, PL: 30* está en el penúltimo puesto. La configuración *CR: 1.0, MR: 0.1, ISL: 0.3, PL: 30*, también está en los últimos puestos (duodécimo).

Sabiendo esta información y analizando la Figura 6.3, podemos apreciar que esas dos configuraciones son las que han experimentado una evolución más rápida en la cantidad de genes o palabras seleccionadas y, a pesar de tener una buena puntuación *fitness*, resultan ser malas clasificando el conjunto de validación. Por el contrario, las tres primeras configuraciones de la Tabla 6.3 son las que mejores resultados han obtenido.

¿A qué se puede deber esto? Se debe a que aquellas configuraciones cuyas poblaciones han avanzado muy rápido, probablemente lo hayan hecho añadiendo más cantidad de palabras en lugar de mejorar mezclarse más veces con otros individuos y mejorar el *fitness* con el mismo número de palabras seleccionadas. Por lo tanto, concluimos que, las configuraciones que han tenido una evolución más lenta, probablemente aporten mejores soluciones, pues los individuos experimentan mejoras mezclando información genética con otros individuos.

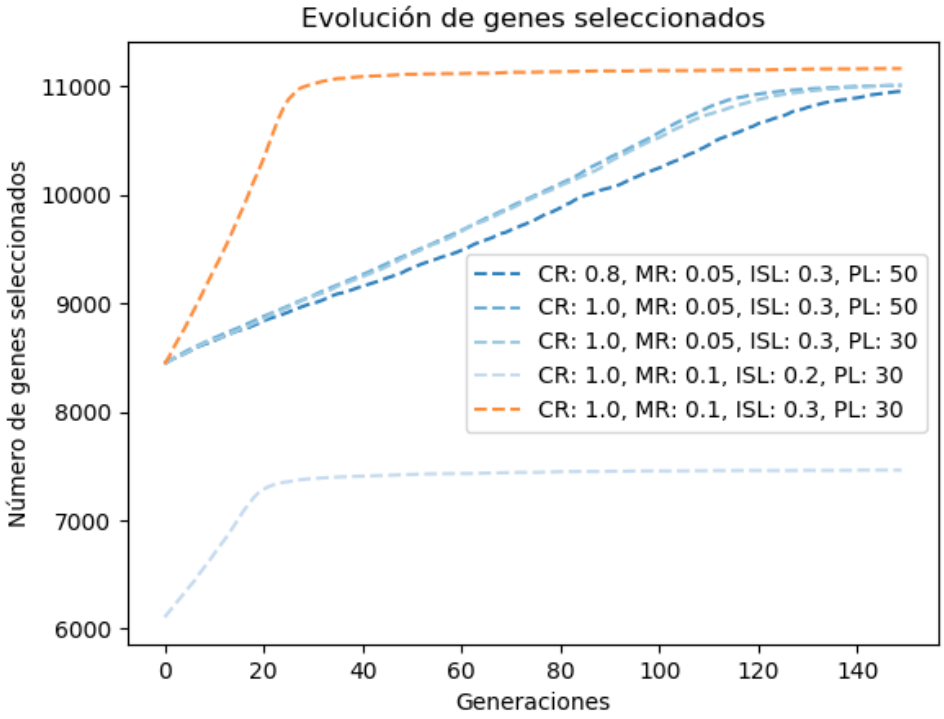


Figura 6.3: Crecimiento medio del número total de palabras que representan a las categorías.

## 6.2. Pruebas de clasificación

En esta sección se estudiarán las soluciones aportadas por el algoritmo genético con un conjunto de validación, con el fin de evaluar la calidad real de las soluciones.

En primer lugar describiremos el número de palabras del que consta cada categoría de nuestro conjunto de validación. En total suman 28.907 palabras que se reparten de la siguiente manera:

- **Religión (soc.religion.christian)**: contiene 10.169 palabras únicas después del proceso de limpieza.
- **Armas (talk.politics.guns)**: contiene 7.052 palabras únicas después del proceso de limpieza.
- **Oriente Medio (talk.politics.mideast)**: contiene 11.686 palabras únicas después del proceso de limpieza.

Para evaluar la calidad del conjunto de palabras que el algoritmo genético seleccionó para representar a cada una de las categorías, se usaron los mismos clasificadores que se utilizaron para la función de evaluación, sin embargo, el conjunto de datos ahora es un conjunto totalmente nuevo.

En la Tabla 6.3 se muestran los resultados ordenados según el valor-F [31], cuya fórmula es la que se puede ver en (6.1).

$$valor\text{-}F = 2 * \left( \frac{precision * recall}{precision + recall} \right) \tag{6.1}$$

| Configuración | Valor-F | Precisión media | Recall medio | Tiempo medio (s) |
|---|---|---|---|---|
| CR: 1.0, MR: 0.05, ISL: 0.3, PL: 50 | 0.7203 | 0.7199 | 0.7261 | 4662.8 |
| CR: 0.8, MR: 0.05, ISL: 0.3, PL: 50 | 0.7148 | 0.7151 | 0.721 | 4214.4 |
| CR: 1.0, MR: 0.05, ISL: 0.3, PL: 30 | 0.7121 | 0.7126 | 0.716 | 3413.4 |
| CR: 1.0, MR: 0.05, ISL: 0.2, PL: 30 | 0.7104 | 0.7111 | 0.7131 | 1968.1 |
| CR: 1.0, MR: 0.05, ISL: 0.2, PL: 50 | 0.7084 | 0.7089 | 0.7145 | 4284.7 |
| CR: 0.8, MR: 0.05, ISL: 0.3, PL: 30 | 0.7082 | 0.7082 | 0.7102 | 2993.0 |
| CR: 0.8, MR: 0.05, ISL: 0.2, PL: 50 | 0.7015 | 0.7014 | 0.7063 | 2532.7 |
| CR: 0.8, MR: 0.05, ISL: 0.2, PL: 30 | 0.6998 | 0.6998 | 0.7054 | 1691.8 |
| CR: 1.0, MR: 0.1, ISL: 0.3, PL: 50 | 0.6895 | 0.6899 | 0.695 | 4824.8 |
| CR: 0.8, MR: 0.1, ISL: 0.2, PL: 50 | 0.6851 | 0.6855 | 0.689 | 2752.5 |
| CR: 0.8, MR: 0.1, ISL: 0.3, PL: 50 | 0.6837 | 0.6846 | 0.6881 | 4882.1 |
| CR: 1.0, MR: 0.1, ISL: 0.3, PL: 30 | 0.6799 | 0.6804 | 0.6887 | 3578.1 |
| CR: 1.0, MR: 0.1, ISL: 0.2, PL: 50 | 0.6796 | 0.6805 | 0.6869 | 2870.1 |
| CR: 0.8, MR: 0.1, ISL: 0.3, PL: 30 | 0.6779 | 0.6784 | 0.6832 | 3642.7 |
| CR: 1.0, MR: 0.1, ISL: 0.2, PL: 30 | 0.6628 | 0.6625 | 0.6703 | 2001.1 |
| CR: 0.8, MR: 0.1, ISL: 0.2, PL: 30 | 0.6605 | 0.6603 | 0.6662 | 2965.5 |

Tabla 6.3: Resultados de la clasificación con cada configuración haciendo la media entre las tres categorías y las diez ejecuciones.

En todos los casos se ha cogido al mejor individuo de la ultima generación y, con el conjunto de validación, se han sacado las anteriores métricas haciendo la media para cada categoría, pues al haber tres cada una tiene su propio valor-F, precisión y recall.

Como se puede apreciar, los resultados están bastante próximos, sin embargo hay una configuración que destaca sobre las demás por los buenos resultados obtenidos y por el tiempo medio que ha tardado en conseguir esa solución. Nos referimos a la configuración *CR: 1.0, MR: 0.05, ISL: 0.2, PL: 30*, en la que además apreciamos que se ha logrado con el menor porcentaje de genes seleccionados, y la población más pequeña.

En las 160 ejecuciones que se realizaron, el mejor resultado obtenido haciendo la media del Valor-F para cada una de las categorías, fue con la siguiente configuración:

- **Tasa de cruce**: 100 %
- **Tasa de mutación**: 5 %
- **Porcentaje de genes seleccionados**: 30 %
- **Tamaño de la población**: 50 individuos

Los resultados que se obtuvieron fueron los que se pueden ver en la Tabla 6.4.

| Categorías | Valor-F | Recall | Precisión | Tiempo |
|---|---|---|---|---|
| Religión (soc.religion.christian) | 0.7999 | 0.8391 | 0.7643 | 5808 |
| Armas (talk.politics.guns) | 0.7579 | 0.7225 | 0.7969 | |
| Oriente Medio (talk.politics.mideast) | 0.7443 | 0.7393 | 0.7493 | |

Tabla 6.4: Métricas obtenidas con el mejor resultado obtenido con la media del Valor-F.

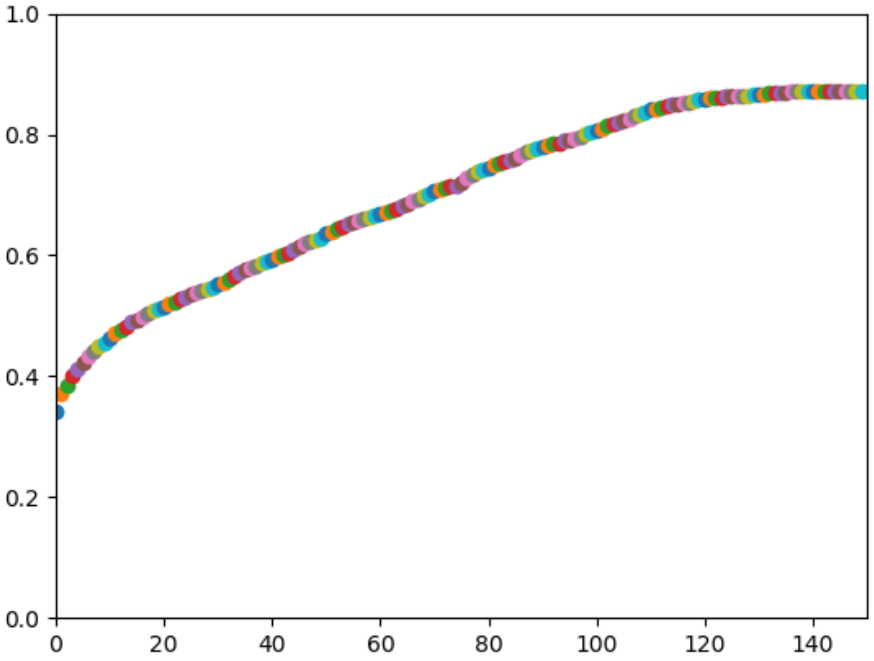


Figura 6.4: Evolución media de la puntuación con la que se obtuvieron los mejores resultados.

Como se puede apreciar en la Figura 6.4, a lo largo de las 150 generaciones se produjo una evolución moderada pero continua de la población. Los resultados obtenidos con un 30 % del total del conjunto de palabras o genes han sido buenos. Pero, pongámoslo en contexto con los obtenidos en el mejor de los casos. Este se da cuando usamos el 100 % de las palabras que hay en cada una de las categorías. Ver Tabla 6.5.

| **Categorías** | **Valor-F** | **Recall** | **Precisión** |
|---|---|---|---|
| Religión (soc.religion.christian) | 0.9 | 0.93 | 0.9 |
| Armas (talk.politics.guns) | 0.9 | 0.97 | 0.9 |
| Oriente Medio (talk.politics.mideast) | 0.86 | 0.76 | 0.86 |

Tabla 6.5: Resultados de la clasificación usando todo el conjunto de entrenamiento.

Evidentemente, los resultados que se observan en la Tabla 6.5 son mejores. Esto se debe a que se usa una mayor cantidad de palabras, y evidentemente, esto mejora los resultados. Sin embargo, lo destacable de los resultados obtenidos con el algoritmo genético es que tan sólo con un 30 % se pueden obtener unos resultados nada desdeñables.

# Capítulo 7

# Conclusiones y líneas futuras

En este capítulo se expondrán las conclusiones extraídas del trabajo y las líneas futuras del mismo.

## 7.1. Conclusiones

El desarrollo de este proyecto ha servido para estudiar muchas técnicas y para profundizar más aún los conocimientos en algoritmos evolutivos y más concretamente en algoritmos genéticos. En un principio no se pensó en utilizar la computación evolutiva para dar una solución al problema trabajado, sin embargo, creo que ha sido una decisión acertada el haberlo hecho, pues considero que la mayoría de veces estos problemas se afrontan usando otras técnicas y considero que es una manera diferente, original y efectiva de llevar a la cabo clasificación de textos.

El haber elegido *Python* como lenguaje de programación no fue una decisión arbitraria, desde un principio, y aún no habiendo trabajado con él previamente, decidimos usarlo debido a la gran cantidad de herramientas y bibliotecas relacionadas con el *machine learning* y la inteligencia artificial. Además, su flexibilidad y rapidez para implementar de forma rápida hace que se puedan experimentar diferentes formas de abordar un problema antes de dar con la mejor de ellas.

Otra de las conclusiones que se extrae confirma, que un buen diseño previo, facilitará mucho el desarrollo del proyecto. En un inicio se había tomado un camino que no nos llevaría a obtener los resultados esperados. En ese entonces la aplicación ya estaba acabada, sin embargo, cuando nos percatamos de que no tenía tanto sentido abordarlo de esa manera, se tuvo que cambiar gran parte del código de la aplicación. Esto se debió a que en ese momento el código no era tan modular como ahora. Al final, resultó complicado adaptar ese código a la nueva forma de abordar el problema, así que se decide iniciar el diseño de la aplicación. Esta tarea se prolongó durante varios días, sin embargo, con los conocimientos que había adquirido no resultó tan complicado como la primera vez. Finalmente, se terminó por implementar el diseño que se había pensado y, aunque podría ser más modular, me ha permitido probar con diferentes configuraciones y parámetros de manera mucho más rápida y sencilla.

La dificultad para encontrar un conjunto de datos de calidad es muy alta, de hecho llevó bastante tiempo encontrarlo, pues en un principio, con la idea de que la empresa finalmente colaborara, se buscó un conjunto de datos lo más similar posible al inicial, lo cuál resultó ser una tarea extremadamente compleja, no solo por el hecho de encontrar

un conjunto de datos en español (que esto no es del todo complicado) sino por el hecho de que su estructura se asemejara a las de los correos electrónicos y que además estuviera preparado para aplicar técnicas de aprendizaje supervisado.

Al principio se encontraron dificultades a la hora de determinar la función de evaluación para el algoritmo genético. En la primera iteración, se pensó en contar el número de veces que aparecían las palabras que aportaba un individuo en el conjunto de entrenamiento, teniendo en cuenta la longitud del conjunto de correos para una determinada categoría, sin embargo, esto no dio los frutos que se esperaban y fue en ese entonces cuando se pensó en usar un clasificador.

Una de las principales preocupaciones a la hora de utilizar un clasificador como evaluador, era que tardara demasiado en retornarnos un resultado con el que poder evaluar la calidad de nuestro individuo, pero estas dudas quedaron despejadas en cuanto se implementó de esta manera y se vió que, efectivamente, era viable.

## 7.2. Líneas futuras

En esta sección se detallarán algunos puntos de mejora y tareas que podrían ser realizadas con el fin de mejorar los resultados y el funcionamiento general del algoritmo genético.

- En primer lugar, se podría aplicar una refactorización al código, si bien hay muchas partes que se encuentran bien diseñadas y con una buena estructura modular, hay otras, que se pueden mejorar o cuya funcionalidad se puede ampliar y generalizar. Es el caso del módulo de *Utilidades*, un módulo que contiene pocas clases y no demasiadas funcionalidades. Podría ser extendido y usar métodos para que se pueda usar por cualquier otra clase de otro módulo, como sí que pasa con el módulo de *Log*, encargado de mostrar los mensajes por pantalla del proceso de ejecución.
- Otra de las cuestiones que se pueden abordar es probar con otras operaciones de cruce, mutación, selección de padres, estrategia de supervivencia o función de evaluación. Por cuestiones de tiempo, no se han podido implementar una variedad amplia de ellas. No obstante, tal y cómo está la aplicación diseñada (sobre todo el módulo genético), no resultaría especialmente complicado implementar otras operaciones y probar con ellas.
- Las pruebas es otro de los aspectos en los que se puede profundizar más, también por cuestiones de tiempo, no se pudo profundizar todo lo deseado y no se pudieron hacer todas las pruebas que en un principio se planearon.
- La documentación es uno de los puntos pendientes de la aplicación, no hay documentación para la aplicación, aunque siempre se intentó de escribir el código de manera comprensible para una persona que desconociera el funcionamiento general de la misma, no le fuera demasiado complicado entenderlo.
- Entender y profundizar en el funcionamiento de los clasificadores empleados ya que no se pudo profundizar todo lo que se habría deseado.

- Ante un caso real de clasificación de correos electrónicos, probablemente se pudieran obtener una mayor cantidad de metadatos, lo cuál mejoraría significativamente los resultados.
- En último lugar, podríamos haber dado un mayor peso a la dirección de correo electrónico. Precisamente por lo que hablábamos en la Sección 2.1. Pues al fin y al cabo, lo más probable es que si los correos provenientes de una dirección de correo son, habitualmente, de una categoría, esta se mantenga en futuros correos provenientes de la misma dirección.

# Capítulo 8

# Conclusions and future work

In this chapter, we are going to present the work conclusions and the next tasks we can engage.

## 8.1. Conclusions

During the work development, I have learned to use a lot of techniques to solve the problem and I could go in depth using of evolutive algorithms and particular, in genetic algorithms. When I started to work on the TFG, my first thought was not work with evolutive computation, but I think that I took a good decision, because nowadays, the most of times this problem is not solved with this techniques and for that reason I think the way to solve this problem is different, original and allows us to get our objective: classify text.

When I decided to code the project in Python [21], it was not an arbitrary decision. Initially, it was a little bit difficult to become familiar with it, but again, I think it was correct because Python has a lot of libraries and tools related to machine learning and Artificial Intelligence. Additionally, the flexibility and the speed we can build a program, allow us to test the different ways to solve the same problem and find the best of them.

Another conclusion that I think is proved, is related to the initial design of the application. A good initial design will enable us to do a better application. When we started to code it, I took the wrong way to solve the problem, and when I realized that, I had some issues when I was trying to adjust the code, so I decided to do a good and modular design and then, start to code it again. The fact that code was done from the scratch enables me to configure each algorithm param faster and easier than before.

To find a quality dataset was difficult, because we thought the company would collaborate with us, so we decided to find a similar dataset because the company was going to offer us her Spanish emails dataset with some fields related to email metadata. The truth is that difficult increased because we were going to use supervised learning, and most of the datasets were ready to use with the unsupervised techniques.

When we started to work, we found some difficulties in how we can evaluate the solution quality. Initially, we thought to count the frequency that words, stored in an individual, appear in our train dataset, taking in account the number of emails for each category. But we realized that it wasn’t working fine, so we started to think in use a classifier.

The time that classifier tooks to return us a result was one of the main concerns. But when we started to implement it, we realized that it was feasible.

## 8.2. Future work

In this section, we are going to name some of the improvement points and tasks we could do in order to achieve better results and a better algorithm behaviour.

- Firstly, we can apply code refactorization. There are several good designed and modular classes, but some of them could be refactorized to use it in other application components. For example, the *Utilities* module contains a few classes and not too functionalities, so the module functionalities could be extended in order to use it more frequently, like *Log* module.
- Implement different crossover, mutation, parent selector, survivor strategies or fitness function could be other issues that we could work on. Due time restrictions I could not implement and test a wide variety of operations, but it would not be difficult, because *Genetic* module is ready to work with new class definitions.
- Tests are another one issues that I could deepen, but due time restrictions, we could not run all test that we planned.
- Also, the code documentation is one of the pending tasks because, the application is not documented, although, we try to write the code the best way I know in order to help another person read it.
- Understand and get more familiar with classifier working, because I could not go in-depth all I wanted.
- If we were processing real emails, probably, we would get more metadata, and this could improve our results.
- Lastly, maybe we could give more emphasis to the email address field, because, as mentioned above in Section 2.1, if an email address sends us an email that is in one category, probably, if we receive an email from the same address, the email category will be the same..

# Capítulo 9

# Presupuesto

En este capitulo se detallan los costes de implementación del proyecto. En el Cuadro 9.1 figuran los costes derivados de la implementación del sistema, mientras que en el Cuadro 9.2 se evalúan los costes que supusieron la ejecución de las pruebas.

| **Actividad** | **Horas** | **Coste por horas** | **Coste** |
|---|---|---|---|
| Búsqueda de antecedentes | 40 | 15 € | 600 € |
| Estudio del estado actual | 30 | 15 € | 450 € |
| Búsqueda del conjunto de datos | 25 | 15 € | 375 € |
| Desarrollo del sistema | 60 | 17 € | 1020 € |
| Evaluación de soluciones | 20 | 17 € | 340 € |
| Total | 175 | - | 3385 € |

Tabla 9.1: Costes de desarrollo del proyecto.

| **Actividad** | **Horas** | **Coste por horas** | **Coste** |
|---|---|---|---|
| Desarrollo script para las pruebas | 10 | 15 € | 150 € |
| Ejecución de las pruebas | 72 | 10 € | 720 € |
| Análisis de los resultados | 15 | 15 € | 225 € |
| Total | 97 | - | 1095 € |

Tabla 9.2: Costes de la ejecución y análisis de pruebas.

Los costes totales teniendo en cuenta los costes de desarrollo del proyecto y los costes de ejecución y análisis de pruebas son los que figuran en el Cuadro 9.3

| **Actividad** | **Total de horas** | **Coste** |
|---|---|---|
| Costes de implementación | 175 | 3385 € |
| Costes de ejecución de pruebas | 97 | 1095 € |
| Total | 272 | 4480 € |

Tabla 9.3: Tabla de costes de totales del proyecto.

# Apéndice A

## Apéndice

### A.1. Formato correo electrónico que proporcionaría la consultoría


```
{
        "important": true,
        "tags": [
                "impuestos",
                "envio de documentación"
        ],
        "contents": {
                "content": "<div>Hello World <br>This is an interesting..."
        },
        "header": {
                "headerContent": {
                        "Delivered-To": "rebecca@zylker.com",
                        "Return-Path": "<noreply@email.com>",
                        "Received": "from mail.email.com by mx.email.com",
                        "Date": "Tue, 12 Dec 2017 02:57:08 -0800",
                        "From": "noreply@email.com",
                        "To": "carol@zylker.com",
                        "Message-ID": "<160f017f.1703.-83383867794@email.com>",
                        "Subject": "sales@zylker.com - Email held for Moderation",
                        "MIME-Version": "1.0",
                        "Content-Type": "multipart/mixed; ",
                        "User-Agent": "Email Mail",
                        "X-Mailer": "Email Mail",
                        "messageId ": "5164940000000618000"
                }

        }
}
```

## A.2. Formato de los correos electrónicos provenientes de *Sklearn*

```
From: manes@magpie.linknet.com (Steve Manes)
Subject: Re: Gun Control (was Re: We're Mad as Hell at the TV News)
Organization: Manes and Associates, NYC
Lines: 3

Others dispute that, like Richard Hofstadter, <America As A Gun Culture>,
and Newton and Zimring's <Firearms and Violence in American Life>.  But,
again, statistics between too dissimilar cultures are difficult to
quantify.
```

## A.3. Formato correo electrónico

```
{
    "category": "talk.politics.guns",
    "from_": "amirza@bronze.ucs.indiana.edu",
    "subject": "gun like american express card",
    "organization": "Indiana University",
    "msg": "ha good far admitting doesnt..."
}
```

## A.4. Formato de los datos de entrada

```
{
     "soc.religion.christian": [
          {
               "category": "soc.religion.christian",
               "from_": "by028@cleveland.freenet.edu",
               "subject": "pantheism environmentalism",
               "organization": "Case Western Reserve University, Cleveland...",
               "msg": "pantheism ive debated quite bit think..."
          }
    ]
}
```

## A.5. Función para limpiar cadenas

```
def cleanWord(content: str) -> str:
    content = re.sub(r'[^\w\s]|\w*\d\w*', '', content)
    tokens = nltk.word_tokenize(content)
    tokens = [word.lower() for word in tokens if word.isalpha()
                                                and len(word) > 2]
```

```
    for i in range(len(tokens)):
        tokens[i] = LEMMATIZER.lemmatize(tokens[i],
                pos=__wordCategory__(pos_tag(tokens[i])))

    stop_words_removed = [word for word in tokens if word not in STOP_WORDS]

    return " ".join(stop_words_removed)

def __wordCategory__(penntag):
    morphy_tag = {'NN':'n', 'JJ':'a',
                  'VB':'v', 'RB':'r'}
    try:
        return morphy_tag[penntag[:2]]
    except:
        return 'n'
```

## A.6. Diccionario para la configuración del algoritmo genético

```
config: dict = {
    "crossover_prob": 0.8,
    "mutation_prob": 0.1,
    "individual_max__len": 0.3,
    "population_size": 30,
    "penalty_rate": 0.5,
    "max_generations": 150,
    "chromosome": "BaseChromosome",
    "gen": "BaseGen",
    "individual": "BaseIndividual",
    "population_updater": "BestsIndividuals",
    "parent_selector": "Tournament",
    "penalization_function": "PenaltyDistribution",
    "crossover": "UniformCrossover",
    "mutation": "ControlledMutation",
    "population_generator": "ApproximatedSize"
}
```

# Bibliografía


[1] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact." *MIS quarterly*, vol. 36, no. 4, pp. 303–317, 2012.

[2] Lifewire, "19 fascinating email facts," 2019, [Online] Disponible en https://www.lifewire.com/how-many-emails-are-sent-every-day-1171210.

[3] J. Xu, W. Peng, T. Guanhua, X. Bo, Z. Jun, W. Fangyuan, H. Hongwei *et al.*, "Short text clustering via convolutional neural networks," 2015.

[4] J. Lilleberg, Y. Zhu, and Y. Zhang, "Support vector machines and word2vec for text classification with semantic features," in *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE, 2015, pp. 136–140.

[5] D. Ruano-Ordás, F. Fdez-Riverola, and J. R. Méndez, "Using evolutionary computation for discovering spam patterns from e-mail samples," *Information Processing & Management*, vol. 54, no. 2, pp. 303–317, 2018.

[6] Contribuidores de wikipedia, "Procesamiento de lenguajes naturales — Wikipedia, la enciclopedia libre," 2019, consultado el 31/05/2019. [Online]. Available: https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales

[7] A. C. Vásquez, J. P. Quispe, A. M. Huayna *et al.*, "Procesamiento de lenguaje natural," *Revista de investigación de Sistemas e Informática*, vol. 6, no. 2, pp. 45–54.

[8] W. B. Cavnar, J. M. Trenkle *et al.*, "N-gram-based text categorization," in *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, vol. 161175. Citeseer, 1994.

[9] Boldon James, "Classifying email – outlook lotus notes," 2019, consultado el 02/06/2019. [Online]. Available: https://www.boldonjames.com/classifying-email/

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[11] J. T. Palma Méndez and R. M. Morales, "Inteligencia artificial. técnicas, métodos y aplicaciones," *Mc Graw*, 2008.

[12] R. P. Díez, A. G. Gómez, and N. de Abajo Martínez, *Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva*. Universidad de Oviedo, 2001.

[13] Contribuidores de wikipedia, “Procesamiento de lenguajes naturales — Wikipedia, la enciclopedia libre,” 2019, consultado el 31/05/2019. [Online]. Available: https://es.wikipedia.org/wiki/N-grama

[14] “Librería scikit-learn,” 2019, consultado el 31/05/2019. [Online]. Available: https://scikit-learn.org/stable/

[15] Carnegie Mellon’s School, “Enron dataset,” 2019, consultado el 03/07/2019. [Online]. Available: https://www.cs.cmu.edu/~enron/

[16] Berkeley, University of California, “Labeled enron dataset,” 2019, consultado el 03/07/2019. [Online]. Available: http://bailando.sims.berkeley.edu/enron_email.html

[17] Python, “Página de documentación para la versión 3.7 de python,” 2019, consultado el 31/05/2019. [Online]. Available: https://docs.python.org/3.7/tutorial/datastructures.html#dictionaries

[18] Sklearn, “Linear support vector classification,” 2019, consultado el 02/07/2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

[19] ——, “One-vs-the-rest (ovr) multiclass/multilabel strategy,” 2019, consultado el 02/07/2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html

[20] Pylint, “Geeks for geeks, tournament selection,” 2019, consultado el 02/07/2019. [Online]. Available: https://www.geeksforgeeks.org/tournament-selection-ga/

[21] Python, “Página web oficial,” 2019, consultado el 31/05/2019. [Online]. Available: https://www.python.org/

[22] Git, “Git,” 2019, consultado el 02/07/2019. [Online]. Available: https://git-scm.com/

[23] GitHub, “Github,” 2019, consultado el 02/07/2019. [Online]. Available: https://github.com/

[24] ——, “Github desktop,” 2019, consultado el 04/07/2019. [Online]. Available: https://desktop.github.com/

[25] Electron, “Electron,” https://github.com/electron/electron, 2013, consultado el 29/05/2019.

[26] JetBrains, “Pycharm,” 2019, consultado el 04/07/2019. [Online]. Available: https://www.jetbrains.com/pycharm/

[27] Microsoft, “Visual studio code,” 2019, consultado el 04/07/2019. [Online]. Available: https://code.visualstudio.com/

[28] Steven Bird, Edward Loper, Ewan Klein, “Natural language toolkit,” 2019, consultado el 04/07/2019. [Online]. Available: https://www.nltk.org/

[29] Google, “Kaggle,” 2019, consultado el 04/07/2019. [Online]. Available: https://www.kaggle.com/

[30] matplotlib, “matplotlib,” 2019, consultado el 04/07/2019. [Online]. Available: https://matplotlib.org/

[31] Towards Data Science, Medium, “Accuracy, precision, recall or f1?” 2019, consultado el 03/07/2019. [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9