PROYECTO FIN DE CARRERA

Filtro anti-spam basado en máquinas de vectores soporte (SVM) y su integración en la herramienta MailScanner

Carlos Rubio Prieto
Ingeniería de Telecomunicación

01/09/2012

Desarrollo, entrenamiento y pruebas de un filtro antispam basado en Máquinas de Vectores Soporte (SVM), y su integración dentro de la herramienta de análisis de correo electrónico MailScanner.

ÍNDICE

1 Introducción 4
2 Estado del arte 5
2.1 Problemática del Spam 5
2.2 Conceptos de Spam y Ham 6
2.3 SpamAssassin 7
2.3.1 Métodos de detección de spam en SpamAssassin 8
2.4 MailScanner 10
3 Objetivo del Proyecto 15
3.1 Justificación 15
4 Clasificación de Textos 16
4.1 Introducción 16
4.2 Factores Críticos 16
4.3 Aprendizaje 17
4.4 Funciones Distancia y Riesgo 19
4.4.1 Configuración Binaria 19
4.4.2 Configuración de Clases Múltiples 20
4.4.3 Configuración de Etiquetas Múltiples 21
4.5 Representación de Textos 22
4.5.1 Nivel sub-palabra 22
4.5.2 Nivel palabra 23
4.5.3 Nivel multi-palabra 25
4.5.4 Otros 26
4.6 Selección de Características o Dimensiones (*Features*) 26
4.6.1 Selección de subconjunto de características 28
4.6.2 Construcción de características 29

5 Análisis Bayesiano 31
6 Máquinas de vectores Soporte 34
6.1 SVMs Lineales de Margen Duro 36
6.2 SVMs de Margen Blando 38
6.3 SVMs No Lineales 38
6.4 SVMs Incrementales de Complejidad Controlada. GSVC: Growing Support Vector Classifier 39
7 Desarrollo del Clasificador 42
7.1 Descripción de la Plataforma de Desarrollo 42
7.2 Selección de *Features* 43
7.3 Entrenamiento 45
7.3.1 Preparación de datos de Entrenamiento y Validación 46
7.3.2 Entrenamiento GSVC 47
7.3.3 Validación GSVC 47
7.4 Clasificación (Test) 48
7.4.1 Representación de Correos Electrónicos 48
7.4.2 Clasificación Del Correo 49
7.5 Resultados Experimentales 50
7.5.1 Experimento 1 50
7.5.2 Experimento 2 52
7.5.3 Experimento 3 54
7.5.4 Experimento 4 55
7.5.5 Experimento 5 58
7.6 Integración con MailScanner Y SpamAssassin 60
7.6.1 Análisis de Integración 62
7.6.2 Modificación de MailScanner. 64
7.6.3 Experimento 6 66
8 Conclusiones 68

9 Agradecimientos 70

10 Bibliografía 71

11 Anexo A – Subrutina IsSpam del módulo Message.pm 73

12 Anexo B – Código Fuente Extracción de *Features* 81

13 Anexo C – Código Fuente Generación Ficheros Train y Test ... 90

14 Anexo D – Código Fuente para CustomFunctions de MailScanner 106

15 Anexo E – Código Fuente para el envío automático de correos a través de sendmail 114

1 INTRODUCCIÓN

En este documento se realiza un estudio sobre la problemática del spam en la sociedad actual (Problemática del Spam). Se describen dos de las herramientas más usadas en la lucha contra el spam (SpamAssassin y MailScanner en los apartados 2.3 y 2.4, respectivamente), y se muestra cómo MailScanner hace uso de SpamAssassin para complementar su análisis del correo electrónico.

En apartados posteriores se describe el objetivo del proyecto, consistente en la mejora de las capacidades de detección de correo no deseado de MailScanner y SpamAssassin con el desarrollo de un módulo de detección basado en máquinas de vectores soporte.

Se describirá funcional y técnicamente la solución implementada, así como su integración en MailScanner. Asimismo, se describirán las etapas de entrenamiento de la solución, indicando los parámetros principales que la caracterizan.

A continuación se describirán las pruebas realizadas y se analizarán los resultados en términos de eficacia y eficiencia, demostrándose que el nuevo motor conseguirá una notable mejora del sistema, contribuyendo de esta forma a la lucha contra el spam.

En el apartado de conclusiones tratarán de extraerse los principales parámetros de la solución implementada, y se describirán los pasos futuros para la evolución de la misma.

2 ESTADO DEL ARTE

2.1 PROBLEMÁTICA DEL SPAM

Las comunicaciones comerciales no solicitadas o spam, como se las conoce más comúnmente, se han convertido en una de las mayores plagas que afectan hoy en día el mundo digital. En un periodo muy corto de tiempo, el correo spam ha llegado a ser más numeroso que los correos electrónicos legales, siendo enviados cientos de millones de mensajes de spam al día.

Esto está provocando grandes costes financieros y pérdidas en productividad para los proveedores de servicios, negocios y usuarios finales. Con la creciente dependencia de los usuarios de internet y del correo electrónico para sus comunicaciones personales y profesionales, el fenómeno del spam puede dificultar seriamente el desarrollo de la economía digital y de la sociedad haciendo disminuir la confianza de los usuarios en las actividades comerciales en la red.

Aunque no hay una definición aceptada universalmente, el término se usa normalmente para describir las comunicaciones electrónicas no solicitadas a través del correo electrónico, móviles (SMS, MMS) y los servicios de mensajería instantánea; normalmente con el objetivo de publicitar productos comerciales o servicios. El contenido de estos mensajes puede variar desde anuncios de billetes de avión de bajo coste hasta material pornográfico ofensivo.

Mientras esta descripción cubre la mayoría de los tipos de spam, actualmente está surgiendo otro fenómeno que es el uso del spam como soporte para llevar a cabo actividades fraudulentas y criminales (incluyendo intentos de obtener información financiera, como números de cuenta y passwords) formateando los mensajes como si provinieran realmente de los bancos. Este último fenómeno es conocido como “brand-spoofing” o “phising”).

Los creadores de spam han desarrollado una gran habilidad para evitar su detección, incluyendo la falsificación del origen del correo

electrónico y la aleatorización del contenido del correo para burlar a los filtros de spam. El problema ha crecido a una velocidad tan grande que la mayoría de los países están promulgando leyes antispam, aunque cada uno aporta diferentes soluciones al problema. Al mismo tiempo, cada vez se acepta más que la lucha contra el spam es un asunto en el que se requiere una mayor coordinación y cooperación internacional.

Mientras las grandes compañías disponen de soluciones de correo completas adquiridas y mantenidas por los principales vendors del mercado, que ya incluyen filtros antispam con buenas prestaciones, las pequeñas organizaciones no pueden afrontar esta inversión, y disponen de pequeños servidores de correo, o bien utilizan servicios en la nube, mucho más asequibles. Es en estas pequeñas compañías y organizaciones donde la solución descrita en el presente proyecto encontrará su nicho de mercado.

2.2 Conceptos de Spam y Ham

En el ámbito de la lucha contra el spam, se manejan dos términos para referirse al correo no deseado (Spam) y al correo deseado (Ham).

En cuanto a los parámetros de medición de la eficacia de un detector de spam se manejan varios, aunque los principales, que serán los utilizados a lo largo del documento, son los siguientes:

- Probabilidad de detección de spam: es la proporción de mensajes spam detectados frente al número de mensajes spam analizados. El objetivo de este parámetro es el 100%, que significaría detectar todo el correo spam.

- Probabilidad de falso positivo: es la proporción de mensajes ham identificados como spam sobre el total de mensajes ham analizados. El objetivo de este parámetro es el 0%, que significaría no detectar ningún mensaje como spam de forma errónea.

En la tarea de optimizar las herramientas de detección de spam, el objetivo será maximizar el primero fijando un límite en el segundo. Aunque dependerá del tipo de usuario y del tipo de información intercambiada, el límite de la probabilidad de falso positivo será en general

0. Esto es debido a que para un usuario generalmente tiene mucho más impacto negativo la clasificación errónea como spam de un mensaje ham que la no detección de unos cuantos mensajes spam.

Además, cuanto mejor es el sistema en base a estos dos parámetros, mayor es el impacto negativo de tener un falso positivo, ya que el usuario habrá perdido la costumbre de revisar el buzón de spam para detectar estos casos.

Aunque se hablará de ello más adelante, será necesario evitar el sobreentrenamiento del sistema, ya que afectará a los dos parámetros anteriores de forma radicalmente negativa.

2.3 SpamAssassin

SpamAssassin es un software que analiza mensajes de correo electrónico, estima la probabilidad de que el mensaje sea o no spam, y devuelve sus conclusiones. Se trata de un sistema basado en reglas. Compara las diferentes partes de los correos electrónicos con una gran cantidad de reglas predefinidas. Cada una de estas reglas suma o resta puntos de la puntuación de spam de cada mensaje. Un mensaje con una puntuación suficientemente alta se marca como spam.

Existen muchos sistemas de comprobación de spam disponibles. SpamAssassin se ha hecho popular por varias razones:

- Usa una gran cantidad de reglas de diferentes tipos y le da un peso a cada una de acuerdo a un estudio previo de las mismas. Las reglas que han demostrado ser más efectivas discriminando correos spam de correos no spam tienen asignados pesos más altos.
- Permite configurar fácilmente las puntuaciones asociadas a cada regla y añadir nuevas reglas basadas en expresiones regulares.
- SpamAssassin puede adaptarse a cada entorno, aprendiendo a reconocer emisores fiables y a identificar nuevos tipos de spam.
- Puede enviar spam a diferentes centros de intercambio de información de spam y puede configurarse para crear trampas para el spam.

- Es un software gratuíto, distribuído bajo la Licencia Pública de "GNU" o la "Artistic License". Cualquiera de ellas permite a los usuarios modificar libremente el software y redistribuírlo.

2.3.1 MÉTODOS DE DETECCIÓN DE SPAM EN SPAMASSASSIN

SpamAssassinn analiza los mensajes de varias formas:

- Puede comprobar si las cabeceras del mensaje son consistentes y cumplen los estándares de internet. Por ejemplo, puede comprobar si el formato de fecha es correcto.

- Puede comprobar si en las cabeceras y el cuerpo del mensaje hay palabras o frases típicas del spam, como "MAKE MONEY FAST", en varios idiomas.

- Las cabeceras y el cuerpo del mensaje pueden ser contrastadas con bases de datos en línea que contienen checksums de mensajes de spam verificados.

- Se puede comprobar la dirección IP del emisor del mensaje en listas de IPs que se conoce son usadas por creadores de spam.

- Puede construir listas negras y blancas de direcciones, hosts o dominios.

- SpamAssassin puede ser entrenado para reconocer spam, haciéndolo aprender con un conjunto de mensajes etiquetados como spam y otro conjunto de mensajes etiquetados como no spam (también llamados "ham").

- La versión 3.0 de SpamAssassin permite comparar la IP del emisor con su dominio usando el protocolo SPF (Sender Policy Framework) para determinar si el sistema tiene permiso para enviar mensajes de los usuarios de ese dominio.

Se puede extraer por tanto que SpamAssassin dispone de dos grupos de métodos de detección claramente diferenciados:

1. Modelo estático de evaluación de reglas

Este modelo es en gran medida estático y manual desde el punto de vista de configuración, ya que exige al administrador realizar una tarea periódica de ajuste de parámetros, listas negras y blancas, frases típicas de correos de spam, listas de IPs no seguras, etc.

Este modelo es capaz de realizar un primer filtrado de correo para eliminar aquellos correos que muestran las características más evidentes de spam. Además de observar el cuerpo del mensaje, también aplica reglas sobre las cabeceras, origen, asunto, destinatarios, etc.

En parámetros de eficacia, este modelo es capaz de identificar típicamente hasta alrededor de un 70% del correo spam recibido. Además, las necesidades de computación son bajas, dado que lo único que debe hacer es evaluar reglas sencillas.

Por todo esto, dentro de la solución propuesta, este grupo de métodos de detección de spam será siempre el primero a ejecutar.

2. Modelo dinámico de reconocimiento basado en una máquina de aprendizaje mediante entrenamiento.

Este modelo es mucho más potente que el anterior, ya que es capaz de construir una máquina de clasificación de correos a partir de una configuración inicial más un entrenamiento periódico. Esto último hace que sea un método dinámico, capaz de adaptarse a las variaciones del spam a lo largo del tiempo, y todo esto sin necesidad de interacción humana.

La máquina implementada por SpamAssassin es una máquina de Bayes basada en un cálculo de probabilidades sobre las palabras que forman el cuerpo de los mensajes.

Se analizan en el entrenamiento sendos buzones de spam (correo no deseado) y ham (correo deseado), extrayendo un listado de palabras (el número es configurable) cuyo número de ocurrencias en uno y otro buzón son descriptivos.

Tras el entrenamiento, a la llegada de un nuevo correo, se extraen las palabras que forman el cuerpo y el número de ocurrencias. Se compara con la lista de palabras resultado del entrenamiento, y se calcula con el conjunto la probabilidad de que el correo sea o no spam.

Las necesidades de computación son sensiblemente mayores que en el modelo estático de detección de spam. Sin embargo, la mayoría del tiempo de procesado se centra en los trabajos de entrenamiento, mientras que el procesado de un correo individual puede llevarse a cabo en pocos segundos.

En términos de eficacia, SpamAssassin es capaz de alcanzar más de un 95% de probabilidad de detección de spam, con un 0% de falsos positivos. Estos números muestran la gran potencia de esta herramienta.

2.4 MAILSCANNER

MailScanner es un sistema de seguridad de correo electrónico de libre distribución. Es el sistema más usado en el mundo, y se está convirtiendo en el sistema estándar en muchos ISP's para la protección contra virus y el filtrado de spam.

MailScanner protege contra virus, spam y ataques contra vulnerabilidades de seguridad, y juega un papel muy importante en la seguridad de una red.

MailScanner proporciona un motor que escanea los correos entrantes, y detecta ataques de seguridad, virus y spam. El correo se acepta, se envía a un directorio de entrada y se encola. MailScanner va procesando los mensajes entrantes que se encuentran en esta cola y los reenvía ya limpios a otra cola en un directorio de salida, desde donde se envían normalmente.

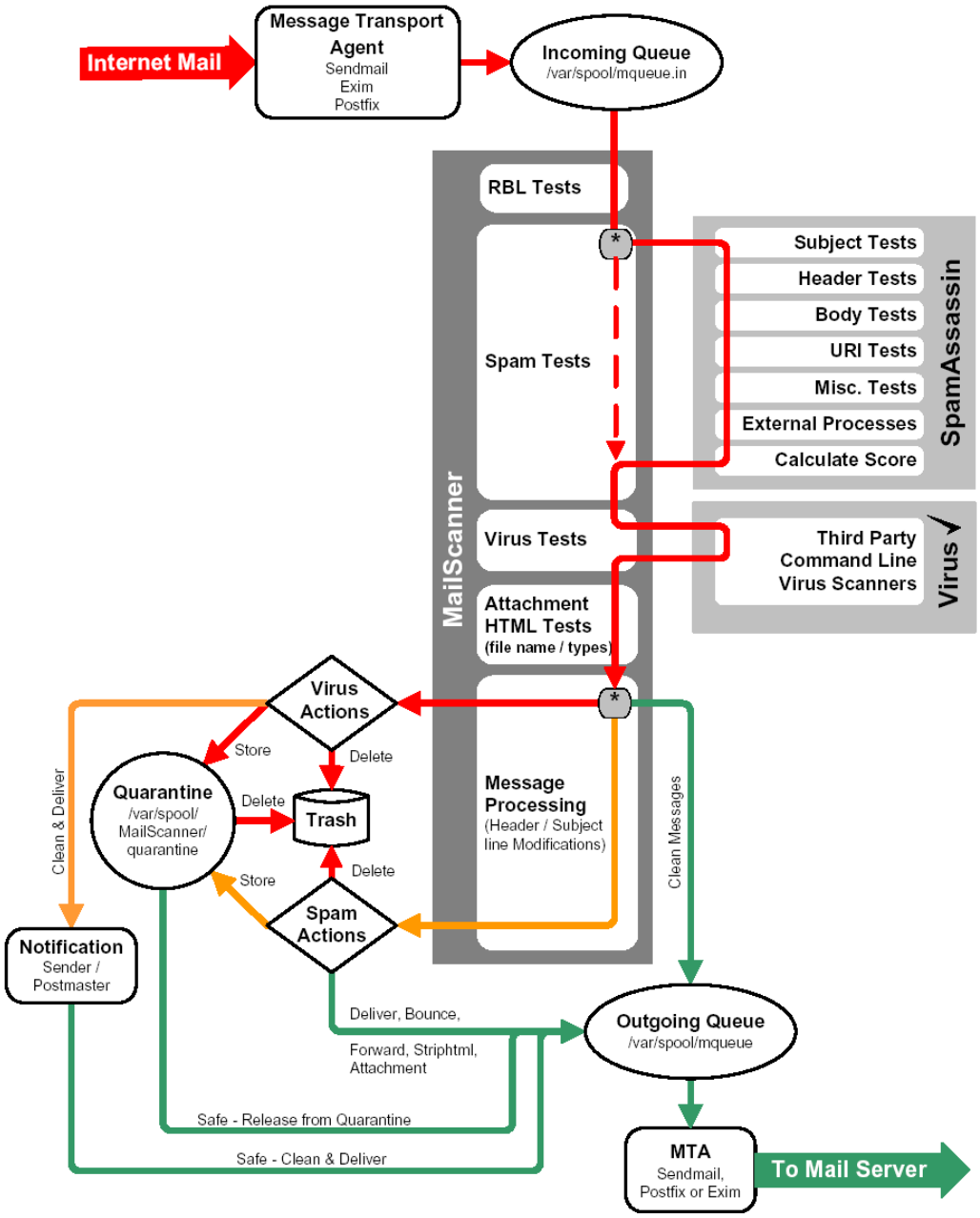
Los mensajes sólo se eliminan de la cola de entrada cuando han sido enviados a la cola de salida, lo que asegura que no haya pérdida de mensajes, incluso ante una pérdida de alimentación en la máquina, ya que siempre habrá una copia de los mensajes que se están procesando.

El motor de MailScanner empieza a escanear los correos iniciando, en la mayoría de sus configuraciones, dos instancias del MTA (Mail Transport Agent). La primera instancia se inicia en modo "demonio" para aceptar todo el correo entrante. El correo se acepta y se envía directamente al directorio de la cola de entrada. La segunda instancia del MTA se inicia

también en modo “demonio” y se encarga de reenviar los correos de la cola de entrada que ya han sido escaneados y procesados.

Para escanear y procesar estos mensajes, se arranca un número configurable de procesos hijos de MailScanner. Normalmente hay cinco procesos hijo que examinan la cola de entrada en intervalos de 5 segundos y seleccionan un número determinado de los correos más antiguos para proceder a su procesado. El número de procesos hijo y el intervalo de tiempo entre ellos se pueden configurar y los valores concretos deberían depender de la velocidad, memoria y número de procesadores de la máquina, así como de la carga que produzcan otras aplicaciones que estén corriendo en ella.

A continuación se muestra el diagrama de flujo del funcionamiento de MailScanner:



Origen: **MailScanner, Open source Secure Mail Gateway Software. Administrators Guide, Version 1.0.1**

ILUSTRACIÓN 1 - FLUJO MAILSCANNER

Típicamente, cuando un proceso hijo encuentra un conjunto de correos entrantes en la cola, si MailScanner está configurado para usar listas negras en tiempo real (Real-time Black-List – RBL): en primer lugar se comprobará si la dirección IP del emisor del correo está en la lista negra.

Si se encuentra en la lista negra, el correo será marcado como spam, y no se harán más comprobaciones, lo que permite ahorrar tiempo.

Si el mensaje pasa la comprobación de la lista negra, será enviado a SpamAssassin, que realizará tests heurísticos, Bayesianos y de otros tipos para determinar el nivel de spam del mensaje. SpamAssassin asigna un valor numérico a cada test que se le hace al mensaje. SpamAssassin también comprueba si el emisor pertenece a una lista negra (spam), o a una lista blanca (no spam). Si pertenece a una lista negra, el valor asignado al mensaje será muy alto, y si pertenece a una lista blanca, el valor será negativo. Cuando SpamAssassin termina de pasar todos los tests, calcula la puntuación total del mensaje.

Se puede configurar MailScanner para que use más de diecisiete antivirus comerciales o de libre distribución. También puede configurarse para que busque virus dentro de ficheros .zip. Si en estos tests se detecta un virus, el mensaje será marcado como infectado.

Cuando la detección de virus termina, el proceso hijo de MailScanner examina los nombres y los tipos de los ficheros adjuntos del correo, evaluando un conjunto de reglas sobre ellos. Dependiendo de cómo se configure MailScanner, cualquier fichero puede ser bloqueado o aceptado.

También se comprueba si el correo contiene código html peligroso como: etiquetas “IFrame”, etiquetas <Form>, o etiquetas <Object Codebase=…>. Según se configure MailScanner, se podrá marcar, pasar, borrar o deshacer estas etiquetas html peligrosas.

Después de todo este proceso, MailScanner tendrá toda la información para modificar, distribuir, rechazar o poner en cuarentena el mensaje, dependiendo del contenido del mensaje y de la configuración.

Si se detecta un virus, MailScanner puede enviar (o no):

- Un mensaje al emisor del virus (normalmente no es apropiado)
- Un mensaje al receptor del virus.
- El mensaje desinfectado al receptor.

- El mensaje y el virus a la cuarentena.

Todos los mensajes tendrán, además, una puntuación de spam (“spam score”). MailScanner podrá configurarse para diferenciar tres tipos de niveles de spam:

- No spam, p. ej. si la puntuación de spam es menor que 6.
- Spam, p. ej. si la puntuación de spam es mayor o igual que 6.
- Spam de alto nivel, p. ej. si la puntuación de spam es mayor que 10.

Para cada uno de los niveles anteriores de spam, MailScanner puede ejecutar cualquier combinación de las siguientes acciones:

- Borrado: borrar el mensaje.
- Almacenamiento: guardar el mensaje en cuarentena.
- Rechazo: enviar un mensaje de rechazo al emisor.
- Reenvío: enviar una copia del mensaje a una dirección de correo.
- Deshacer html: convertir el código html en texto plano.
- Adjunto: convertir el mensaje original en un adjunto del mensaje.
- Entrega: entregar el mensaje normalmente.

3 OBJETIVO DEL PROYECTO

El objetivo principal del proyecto es definir y desarrollar un clasificador de textos alternativo al bayesiano de SpamAssassin, así como su integración en MailScanner.

Como objetivo secundario, este clasificador de textos tratará de mejorar el comportamiento de la herramienta SpamAssassin, haciendo menores las probabilidades de falso negativo y de falso positivo.

3.1 JUSTIFICACIÓN

Como ya se ha comentado anteriormente, un buen filtro antispam debería, por supuesto, detectar todos los correos spam. Pero incluso será más importante minimizar la probabilidad de obtener un falso positivo. Es decir, será muy importante que no deje pasar los correos spam, pero será más importante todavía que no etiquete un correo como spam, cuando en realidad no lo es, ya que esta situación podría provocar, en caso de eliminación del mensaje, una pérdida de información que podría ser crítica.

La herramienta SpamAssassin consigue unas buenas prestaciones en cuanto a la detección de spam. Como se describió en el apartado 2.2., SpamAssassin hace uso de conjuntos de reglas y de un motor Bayesiano para realizar su función.

El motor Bayesiano aprende, mediante un entrenamiento, cómo clasificar un correo (en definitiva, un texto) como "spam" o "no spam" (también llamado "ham"). El entrenamiento se realiza pasándole un conjunto de correos etiquetados, es decir, proporcionándole un conjunto de correos cuya naturaleza es conocida. De esta forma consigue obtener unas reglas basadas en el contenido del texto que le permitirán posteriormente clasificar los correos.

El clasificador desarrollado es un motor basado en máquinas de vectores soporte ("SVM" – Support Vector Machine) que, también mediante entrenamiento, aprenderá a etiquetar los correos como "spam" o "no spam".

4 CLASIFICACIÓN DE TEXTOS

4.1 INTRODUCCIÓN

Con el gran crecimiento de Internet en los últimos años, la tarea de clasificar documentos en lenguaje natural en conjuntos predefinidos de categorías semánticas se ha convertido en uno de los métodos clave para organizar la información en la web. Esta tarea se conoce como clasificación de textos. Su uso puede ir desde la categorización de páginas web por temática, adecuación automática al usuario de los contenidos de páginas web, enrutamiento de mensajes hacia expertos según la temática… Debido a la enorme cantidad de información existente, esta tarea debe ser automatizada.

En este proyecto, se utilizará la clasificación de textos para clasificar semánticamente los correos electrónicos en dos categorías: spam y no spam (o ham).

La construcción de un clasificador de textos automático consiste fundamentalmente en construir reglas de clasificación. En ámbitos sencillos, bastaría con construir dichas reglas de forma manual. Sin embargo, cuando el número de categorías es demasiado grande, o si la definición de las categorías varía en el tiempo, la construcción de las reglas debe estar basada en el aprendizaje máquina.

El aprendizaje máquina está basado en que un algoritmo recibe un conjunto de documentos etiquetados y en base a ellos construye automáticamente reglas de clasificación.

4.2 FACTORES CRÍTICOS

Al enfrentarse a un problema de clasificación de textos, nos enfrentamos a la optimización de los siguientes parámetros:

- Conjunto de datos de entrada muy amplio: La entrada de un clasificador de textos es el lenguaje natural. El lenguaje natural es extremadamente amplio, por lo que la clasificación de textos se enfrenta a un gran número de muestras. Aun

simplificando las transformaciones de las palabras, ignorando su orden y representando los documentos como simples histogramas de frecuencias de aparición de palabras, los datos de entrada pueden tener 30000 dimensiones o más.

- Conjunto de datos de entrenamiento pequeño: Normalmente, los algoritmos de aprendizaje requieren un número de muestras para entrenamiento proporcional al número de dimensiones del conjunto de datos de entrada para producir reglas de clasificación lo suficientemente precisas. En general, suele ser necesario tener un número de muestras para entrenamiento de 50 a 100 veces el número de dimensiones. Sin embargo, en la clasificación de textos tendremos normalmente menos muestras de entrenamiento que dimensiones.

- Ruido: Se considera ruido en la clasificación de textos tanto a los errores ortográficos como a los errores de etiquetado de las muestras de entrenamiento. El clasificador de textos deberá ser capaz de independizar los resultados del ruido.

- Tareas de aprendizaje complejas: La clasificación de textos está basada en el entendimiento semántico del lenguaje natural por un humano. No existen por tanto definiciones formales ni operacionales de las clasificaciones.

- Eficiencia computacional: El entrenamiento del clasificador de textos deberá ser capaz de tratar miles de muestras, y analizar miles de dimensiones, por lo que la eficiencia de computación se convierte en un factor crítico de éxito.

4.3 APRENDIZAJE

El primer problema al que nos enfrentamos en el aprendizaje de los clasificadores de texto es su definición formal. Existen dos líneas de estudio al respecto, la inductiva y la inferencia transductiva. La mayoría de los métodos actuales siguen la línea inductiva, que trata de extraer reglas de clasificación a partir de documentos de entrenamiento, con el fin de lograr

clasificar nuevos ejemplos con una alta precisión. Éste será el método que desarrollaré a lo largo de este proyecto.

Supondremos que las muestras de entrenamiento son independientes y distribuidas idénticamente, y que siguen una distribución fija aunque desconocida. Aunque existen numerosas formas de representar formalmente el problema, a lo largo del proyecto representaré las muestras etiquetadas (datos de entrenamiento) y su distribución de la siguiente forma, ya que es la más adecuada para representar el problema de clasificación de correos electrónicos en dos clases (spam y ham):

$$(\vec{x}_1, y_1),...,(\vec{x}_n, y_n) \rightarrow \Pr(\vec{x}, y)$$

Cada muestra de entrenamiento (cada correo) será representado por el vector $\vec{x}$, y su etiqueta (clase conocida a la que pertenece, spam o ham) se representará como y.

Normalmente en la clasificación de textos, $\vec{x}$ será un vector con un alto número de dimensiones, que representará las palabras que componen el texto.

Una vez representadas las muestras de entrenamiento, será necesario elegir una función que represente el comportamiento del aprendizaje. Para ello, se representarán las reglas de clasificación como $h(\vec{x})$, la función distancia como $L(h(\vec{x}), y) \in \Re$, y la función riesgo como $R(h)$, de forma que:

$$R(h) = \int L(h(\vec{x}), y) d\Pr(\vec{x}, y)$$

$h(\vec{x})$: regla de clasificación, que predecirá la clase a la que pertenece el la muestra $\vec{x}$.

$L(h(\vec{x}), y)$: función distancia, que mide cuán lejos está la predicción de la regla de clasificación $h(\vec{x})$ con respecto a la clase conocida de la muestra de entrenamiento, y.

$R(h)$: función riesgo, que mide la esperanza de la función distancia con respecto a la distribución de las muestras de entrenamiento $\Pr(\vec{x}, y)$.

Por lo tanto, el objetivo del aprendizaje será encontrar la regla de clasificación $h(\vec{x})$ que minimice la función riesgo, $R(h)$.

Como la distribución de las muestras no es conocida, el algoritmo de aprendizaje A deberá encontrar la regla de clasificación h_{A} a partir del conjunto de muestras, S:

$$h_{\mathrm{A}} = \mathrm{A}(S)$$

Por tanto, será necesario encontrar la función distancia y el algoritmo de aprendizaje que optimicen la búsqueda de reglas de clasificación para el problema de clasificar correos en spam y ham.

4.4 Funciones Distancia y Riesgo

Para encontrar funciones distancia y riesgo adecuadas, habrá que tener en cuenta las características del problema a resolver. A continuación haré una breve descripción de las opciones disponibles atendiendo al rango de valores de las clases en la clasificación: binaria, múltiples clases y múltiples etiquetas. Para el problema que nos ocupa de clasificación de correos, la opción que mejor se ajusta es la binaria, puesto que necesitamos únicamente conocer si un correo electrónico es spam o no. Por ello, se describirá esta configuración binaria con más detalle que las demás.

4.4.1 Configuración Binaria

La configuración binaria es la más sencilla, aunque también es la más importante en el aprendizaje. Otras configuraciones pueden ser reducidas a una binaria teniendo en cuenta algunas restricciones. En la configuración binaria tendremos exactamente dos clases. A priori, es la configuración que más se ajusta al problema de clasificar correos electrónicos en exactamente dos clases, spam y ham. Por lo tanto las etiquetas de las muestras de entrenamiento tendrán dos valores posibles. La forma de representarlo será la siguiente:

$$y \in \{-1,+1\}$$

La función distancia más comúnmente utilizada en la configuración binaria es la distancia 0/1, que hará que la distancia sea 0 cuando el resultado de aplicar la regla de clasificación sea igual a la etiqueta de la

muestra, y 1 cuando estos valores no sean iguales. La función distancia se representa como sigue:

$$L_{0/1}(h(\vec{x}), y) = \begin{Bmatrix} 0 & h(\vec{x}) = y \\ 1 & en \quad otro \quad caso \end{Bmatrix}$$

La función riesgo asociada a esta función distancia es la tasa de error, o $Err(h)$, que representará la probabilidad de realizar una predicción falsa ante una muestra de entrenamiento aleatoria, que sigue la distribución $\Pr(\vec{x}, y)$.

$$Err(h) = \Pr(h(\vec{x}) \neq y \mid h) = \int L_{0/1}(h(\vec{x}), y) d \Pr(\vec{x}, y)$$

La tasa de error descrita trata a todos los errores por igual. Como ya se ha comentado anteriormente, en el problema de clasificación de correos como spam/ham no todos los errores tienen el mismo impacto, ya que eliminar un correo importante por considerarlo spam tiene un impacto negativo mayor que clasificar un correo spam como ham. Para tener en cuenta esto, podrían emplearse factores de coste en la función distancia, de manera que cada tipo de error en la clasificación tuviera un impacto distinto:

$$L_{0/1}(h(\vec{x}), y) = \begin{Bmatrix} C_{+-} & h(\vec{x}) = +1; & y = -1 \\ C_{-+} & h(\vec{x}) = -1; & y = +1 \\ 0 & otro & caso \end{Bmatrix}$$

En este caso, se introducen otros dos parámetros a ser optimizados: los costes de error de clasificación, lo que hace más complejo aun el algoritmo. De cara a desarrollar el entrenamiento para el clasificador de spam objeto del proyecto, se considerarán las dos posibles funciones distancia, y se elegirá la más adecuada atendiendo a parámetros de precisión y de coste computacional.

4.4.2 Configuración de Clases Múltiples

En algunos problemas de clasificación, el número de clases puede ser mayor que dos. Por ejemplo para conocer la temática de un artículo de investigación. En general se representa el conjunto de clases de la siguiente forma:

$$y \in \{1, \ldots, l\}$$

, donde l es el número de clases.

En este caso, las funciones distancia y riesgo (tasa de error) descritas en el caso binario son también válidas. Asimismo, también son válidas las consideraciones acerca de los factores de coste de cada error.

El coste computacional de esta configuración será siempre más alto que en el caso binario. Sin embargo, es posible reducir el problema a l problemas binarios, con lo que el coste computacional se reduce drásticamente.

4.4.3 Configuración de Etiquetas Múltiples

Aunque no es el caso en la clasificación de correos electrónicos como spam/ham, en la mayoría de los problemas de clasificación de textos nos encontramos con etiquetas múltiples. Esto significa que existen varios conjuntos de etiquetas no necesariamente disjuntos, por lo que cada documento debe ser clasificado en cada uno de los conjuntos, pudiendo clasificarse en uno, en varios, o en ninguno de los conjuntos de etiquetas o clases.

Suele representarse el conjunto de clases como un vector binario de l dimensiones, donde l es el número total de clases existentes, de la forma siguiente:

$$y \in \{+1, -1\}^l$$

Por tanto, la regla de clasificación debería obtener también un vector binario de l dimensiones.

Este problema es mucho más complejo que los dos anteriores, ya que no hay una forma tan precisa de medir la distancia. En los estudios sobre el tema, se ha encontrado que la distancia de Hamming es razonablemente buena en estos problemas, y se representa de la siguiente forma:

$$L_{Ham\min g}\left(h(\vec{x}), \vec{y}\right) = \sum_{i=1}^{l} L_{0/1}\left(h^{(i)}(\vec{x}), y^{(i)}\right)$$

Y la función de riesgo asociada:

$$R(h) = \sum_{i=1}^{l} Err\left(h^{(i)}\right)$$

Al igual que en la configuración de clases múltiples, este problema también puede ser reducido como múltiples problemas binarios. No se profundizará más en el tema en este proyecto, dado que no es el caso que nos ocupa, aunque se anima al lector a profundizar en material existente al respecto.

4.5 Representación de Textos

Una vez definido el problema, y representadas las reglas de clasificación, las funciones distancia y riesgo, sólo queda definir cómo representar textos como vectores $\vec{x}$ que permitan ejecutar algoritmos y realizar cálculos.

Para ello, se describen a continuación tres formas de representar texto, de las cuales se ha escogido la representación a nivel de palabra para desarrollar este proyecto. La elección se hizo a priori por razones de simplicidad y coste computacional, con la idea de volver sobre ello si con este método no era suficiente. Como se demostrará más adelante, el nivel de palabra cubrió las expectativas.

4.5.1 Nivel sub-palabra

La representación a nivel de sub-palabra se basa en que las unidades básicas que representan al texto son fragmentos de palabras.

La más común de las representaciones a nivel de sub-palabra son los n-gramas, que consisten en descomponer las palabras en todos los fragmentos posibles de n letras consecutivas, incluyendo un espacio al principio y al final de la palabra. Por ejemplo, para n=3, los trigramas de la palabra “casa” serían “_ca”, “cas”, “asa” y “sa_”, siendo _ un espacio en blanco.

Esta representación facilita el conteo de palabras similares. Por ejemplo una palabra en singular y en plural comparten la mayoría de sus n-gramas.

Por otro lado, esta misma facilidad hace que se puedan cometer errores, al encontrarnos con palabras que comparten muchos de sus n-gramas, y que sin embargo son radicalmente distintas en significado.

Otra ventaja de esta representación es que es más inmune que las demás a los errores ortográficos.

La representación en n-gramas es sencilla y directa para cualquier idioma, independientemente de la complejidad del mismo.

4.5.2 Nivel palabra

La descomposición a nivel de palabra ha sido y es la más usada en la mayoría de los trabajos de clasificación de textos.

La palabra como unidad básica de representación presenta poca ambigüedad, y aunque existen palabras con varios significados, algunas veces muy distintos, lo cierto y demostrado es que no suelen afectar en demasía a la tarea de clasificación.

La primera ventaja es el coste del algoritmo de descomposición, ya que es relativamente sencillo descomponer un texto en palabras en casi cualquier lenguaje de programación.

El orden de las palabras en el texto, o la estructura del texto en sí generalmente es irrelevante a la hora de clasificar textos, por lo que el resultado de una descomposición de este tipo es un conjunto de palabras con un valor asociado que representa la frecuencia con la que aparece cada palabra en el texto.

Como se ha explicado en apartados anteriores, la forma de representar un documento será un vector con n dimensiones. En este tipo de descomposición, cada palabra representa una dimensión del vector, y el valor del vector en esa dimensión representa de alguna forma la frecuencia de aparición de dicha palabra.

En este proyecto se han utilizado y comparado dos formas de representar la frecuencia de aparición de las palabras, el TF (Term Frequency) y el TFIDF (Term Frequency Inverse Document Frequency). El primero es el más simple de los métodos, y representa el número de ocurrencias de la palabra en concreto en el documento. El segundo, trata de

ser una representación normalizada con la cual poder comparar documentos de una forma más óptima y por tanto trata de mejorar la tareas de entrenamiento y de clasificación.

A continuación se detallarán ambos conceptos, y más adelante se detallará cómo se desarrollaron los algoritmos de entrenamiento y clasificación en base a ellos, y se compararán los resultados para determinar cuál de los dos proporciona mejores resultados.

4.5.2.1 TF (TERM FREQUENCY)

El TF es la medida básica para la atribución de un peso a una palabra en un determinado documento. Representa el número de ocurrencias de una palabra en un documento determinado. Su notación es la siguiente:

$$TF(\omega_i, d_j)$$

Donde ω_i es la palabra en estudio, y d_j es el documento sobre el que se está midiendo el número de ocurrencias de dicha palabra.

La utilidad de esta medida atiende a la observación de que las palabras con más ocurrencias en un determinado documento son más relevantes que las que no lo son.

A continuación se muestra la forma de calcular los pesos de cada palabra en base al TF:

$$x_i = \frac{TF(\omega_i, d)}{\sqrt{\sum_j TF(\omega_j, d)^2}}$$

Como puede observarse, se ha introducido un término de normalización (norma 2), que hace posible comparar documentos independientemente de su tamaño.

Sin embargo, una palabra que sea muy frecuente podría aparecer en casi todos los documentos de un conjunto, por lo que no ayudaría a discriminar entre distintas clases de documentos. Para tener esto en cuenta, aparece la medida Document Frequency (DF), que ayuda a asignar un menor peso a estas palabras. Su notación es la siguiente:

$$DF(\omega_i)$$

Donde ω_i es la palabra en estudio. El DF representa el número de documentos en los que aparece al menos una vez la palabra ω_i. Este parámetro se utilizará en la siguiente medida de peso.

4.5.2.2 TFIDF (TERM FREQUENCY INVERSE DOCUMENT FREQUENCY)

El TFIDF parte de la medida básica TF para asignar pesos a las palabras de un documento. El cambio que introduce es tener en cuenta el DF de cada palabra para ajustar el peso de manera que una palabra con un TF muy alto no tenga un peso muy alto si aparece en la mayoría de los documentos del conjunto de trabajo.

Por tanto, el peso se reducirá de forma proporcional al DF de la palabra en concreto.

A continuación, se muestra cómo se calculará el TFIDF:

$$x_i = \frac{TF(\omega_i, d)\log\left(\frac{|D|}{DF(\omega_i)}\right)}{\sqrt{\sum_j \left[TF(\omega_j, d)\log\left(\frac{|D|}{DF(\omega_i)}\right)\right]^2}}$$

Donde |D| es el número de documentos del conjunto de trabajo (del conjunto de entrenamiento, por ejemplo).

Igual que en el apartado anterior, se ha introducido un término de normalización (norma 2), que hace posible comparar documentos independientemente de su tamaño.

4.5.3 NIVEL MULTI-PALABRA

La descomposición de textos a nivel multi-palabra trata de encontrar grupos de palabras que tienen alguna relación, ya sea sintáctica, semántica o de co-ocurrencia que les hacen ser unidades independientes y representativas del texto bajo estudio.

Cuando se atiende a la relación sintáctica, por ejemplo descomponiendo el texto en sintagmas nominales, la descomposición deja de ser independiente del idioma, lo que añade mucha complejidad a la ya compleja tarea de representación a nivel multi-palabra.

En cualquiera de los casos, el coste computacional de este tipo de descomposición es muy elevado con respecto a los dos niveles anteriores.

4.5.4 Otros

Existen otros niveles de descomposición de textos, que por su complejidad, no merece la pena tenerlos en cuenta para el objetivo del proyecto. Se tratan de la descomposición a nivel semántico y a nivel pragmático. Existen estudios sobre ambos, aunque es muy complicado encontrar una forma automática de descomponer textos tanto en categorías semánticas como pragmáticas. En general siempre se requiere una manualidad.

4.6 Selección de Características o Dimensiones (*Features*)

Una vez definidas las funciones distancia y riesgo, y habiendo elegido un nivel de representación de textos para la clasificación de correos electrónicos, queda afrontar el último paso, la selección de características, o *features* en inglés.

Estas características serán las dimensiones de los vectores que representarán cada correo electrónico, y más concretamente las palabras que lo representarán y a las cuales habrá que dar un valor numérico asociado al número de ocurrencias. Por ejemplo, analicemos el siguiente correo electrónico:

Buenos días,

La semana pasada estuve llamándote, pero no conseguí contactar contigo. Pregunté a tu compañero, Alberto, y me dijo que estabas de vacaciones. ¿Qué tal han ido esas vacaciones? ¿Has estado en la playa? Yo prefiero la playa a la montaña, el calorcito, las cervecitas en el chiringuito… Todo son ventajas.

Por favor, llámame cuando vuelvas de tus vacaciones, que tengo un par de temas que hablar contigo.

Pásalo bien,

Un saludo.

Se listan a continuación las palabras que componen el correo y el número de ocurrencias de cada una:

Buenos	1	Compañero	1	Estado	1	Favor	1
Días	1	Alberto	1	En	2	Llámame	1
La	4	Y	1	Playa	2	Cuando	1
Semana	1	Me	1	Yo	1	Vuelvas	1
Pasada	1	Dijo	1	Prefiero	1	Tus	1
Estuve	1	Que	3	Montaña	1	Tengo	1
Llamándote	1	Estabas	1	El	2	Un	2
Pero	1	De	3	Calorcito	1	Par	1
No	1	Vacaciones	3	Las	1	Temas	1
Conseguí	1	Qué	1	Cervecitas	1	Hablar	1
Contactar	1	Tal	1	chiringuito	1	contigo	1
Contigo	1	Han	1	Todo	1	Pásalo	1
Pregunté	1	Ido	1	Son	1	bien	1
A	2	Esas	1	Ventajas	1	saludo	1
Tu	1	Has	1	Por	1		

De todas las palabras que componen el texto, las que aparecen más de una vez son: la, a, que, de, vacaciones, en, playa, el, un.

Pero surge una pregunta: ¿todas estas palabras son representativas? La respuesta es no. Como era de esperar hay palabras que se repiten en todos los textos, sean de la índole que sean, como son los artículos, preposiciones y pronombres. En el ejemplo anterior, las palabras la, a, que, de, en, el, un, no son representativas, por lo que podrían eliminarse, quedando únicamente las palabras vacaciones y playa como palabras representativas que aparecen más de una vez en el texto.

Con este ejemplo se ilustra una de las primeras tareas que deben ser abordadas en cualquier trabajo de clasificación de textos, la eliminación de palabras irrelevantes o inapropiadas para la clasificación de los textos.

Existen dos razones fundamentales para ejecutar esta eliminación de palabras irrelevantes o stopwords, como se denominan en inglés:

- Evitar el sobreentrenamiento de la máquina. Cuantas más características o dimensiones se manejen, mayor es el riesgo de incurrir en el sobreentrenamiento de la máquina, haciendo que la clasificación sea perfecta con los datos de entrenamiento, y muy mala con datos reales.

- Buscar la eficiencia computacional. A mayor número de dimensiones, mayor es el coste computacional y los requerimientos de memoria de la máquina, algo crítico en la mayoría de los entornos.

Tras la eliminación, o la omisión en el procesado, de las stopwords, será necesario llevar a cabo la selección de características en sí misma.

En la tarea de clasificación de textos es necesario tener un listado de características adecuado para la representación de los textos que se manejen, ya que serán éstas las características que se buscarán en los correos, y no otras. Dado que el lenguaje y las expresiones evolucionan en el tiempo, el listado de características deberá ser actualizado con cierta periodicidad, por ejemplo mensual, o cuando la capacidad de generalización de la máquina se vea afectada.

A continuación se describen dos métodos de selección de características.

4.6.1 Selección de subconjunto de características

Este método trata de obtener un conjunto de características representativo a través de la selección de las mismas a partir del listado completo de palabras del conjunto de entrenamiento.

Para ello se utilizan las siguientes técnicas. Las dos primeras serán utilizadas, al menos en cierta medida, en este proyecto:

- Eliminación de stopwords: como se ha explicado anteriormente, se tratan de eliminar aquellas palabras del lenguaje que son irrelevantes en la clasificación de textos.

- Umbral de frecuencia de documento: trata de establecer un umbral para el número mínimo de documentos en los que debe aparecer una palabra para poder formar parte del conjunto de

características elegido. De esta forma, se eliminarán aquellas palabras que aparecen con muy poca frecuencia. Este método también trata de eliminar el sobreentrenamiento y los problemas de eficiencia computacional.

- Métodos que miden la dependencia o relación de las palabras con la etiqueta del texto, es decir, con el tipo de texto. Con ello, tratan de eliminar aquellas palabras que tienen menor relación con la etiqueta. Algunos de estos métodos son: "Información mutua", "Odds ratio" y "tests χ^2".

El último grupo de métodos hacen cálculos estadísticos basados en ciertas asunciones respecto a la fase de aprendizaje. Por ejemplo, todos ellos asumen hasta cierto punto que las palabras son independientes entre sí, lo cual no es estrictamente cierto.

En este proyecto, utilizaré una técnica basada en los dos primeros métodos, y que se ejecutará sobre los datos de entrenamiento. Para ello, se analizarán los datos de train, extrayendo todas las palabras que los forman, y se eliminarán en primer lugar todas las stopwords que se han considerado oportunas. Tras ello, se calcularán los parámetros Term Frequency (TF) y Term Frequency Inverse Document Frequency (TFIDF), introducidos en apartados anteriores. Dado que en el proyecto se comparará la eficacia de ambos parámetros, se calcularán ambos parámetros para todas las palabras de todos los documentos (correos electrónicos), y por último se ordenarán las palabras de mayor a menor valor en dichos parámetros.

Una vez ordenadas, se seleccionarán las primeras N palabras como *features*, y con ellas se ejecutarán los procesos de entrenamiento y clasificación.

Además de comparar la eficacia de los parámetros TF y TFIDF, se han ejecutado todos los procesos seleccionando distintos valores para N, de forma que se hará una comparación, pudiendo concluirse cuál es el número de *features* que optimiza el entrenamiento y la clasificación.

4.6.2 CONSTRUCCIÓN DE CARACTERÍSTICAS

Las técnicas de construcción de características tratan de agrupar varias palabras en una única característica, atendiendo a rasgos

morfológicos (*Stemming*) o semánticos (*Latent semantic indexing (LSI)*, *Term clustering*), disminuyendo así el número de dimensiones de los vectores, y disminuyendo por tanto la complejidad.

Sin embargo, los métodos de construcción de características son en sí complejos, y necesitan un tiempo elevado de procesado, así como de recursos externos, como *thesauri*.

Además, ambos métodos son dependientes del idioma, por lo que para el problema de clasificación de correos sería necesario disponer de herramientas multiidioma, lo que complicaría demasiado el proceso.

Para la ejecución de este proyecto se ha descartado la construcción de características, ya que se ha considerado suficiente el método de selección de un subconjunto de características.

5 ANÁLISIS BAYESIANO

Como se mencionó en apartados anteriores, SpamAssassin incluye un clasificador de correos Bayesiano para la detección de Spam.

Como se verá en el siguiente capítulo, otros algoritmos de aprendizaje intentan encontrar una cota para una función de error, y conseguir que la probabilidad de superar esa cota sea muy baja. En cambio, el objetivo del análisis bayesiano es encontrar los valores de salida del algoritmo que con mayor verosimilitud estén basados en los datos de entrenamiento. El resultado del entrenamiento será una función con mayor o menor complejidad.

La idea entonces es usar un modelo estadístico para estimar la probabilidad de que un documento d pertenezca a la clase y, o $\Pr(y \mid d)$.

A continuación se describe una forma de estimar $\Pr(y \mid d)$. La regla de Bayes dice que para conseguir la mayor precisión en la clasificación, el documento d debe asignarse a la clase y para la cual $\Pr(y \mid d)$ es la más alta posible:

$$h_{BAYES}(d) = \arg\max_{y \in \{-1,1\}} \Pr(y \mid d)$$

Considerando los documentos por separado según su longitud, l:

$$\Pr(y \mid d) = \sum_{l=1}^{\infty} \Pr(y \mid d, l) \cdot \Pr(l \mid d)$$

$\Pr(l \mid d)$ es igual a 1 para la longitud l' del documento d y 0 en cualquier otro caso. Tras aplicar el teorema de Bayes a $\Pr(y \mid d, l)$ se puede escribir que:

$$\Pr(y \mid d) = \frac{\Pr(d \mid y, l') \cdot \Pr(y \mid l')}{\sum_{y' \in \{-1,+1\}} \Pr(d \mid y', l') \cdot \Pr(y' \mid l')}$$

$\Pr(d \mid y, l')$ es la probabilidad de que el documento d pertenezca a la clase y dada su longitud l'. $\Pr(y \mid l')$ es la probabilidad de que un documento de longitud l' pertenezca a la clase y. A continuación se asume que la categoría de un documento no depende de su longitud, por lo que

$\Pr(y \mid l') = \Pr(y)$. Se puede calcular una estimación de $\Pr(y)$, $\widehat{\Pr}(y)$ de la fracción de documentos que pertenece a la clase y:

$$\widehat{\Pr}(y) = \frac{|y|}{\sum_{y' \in \{-1,+1\}} |y'|} = \frac{|y|}{|D|}$$

$|y|$ es el número de documentos de entrenamiento para la clase $y \in \{-1,+1\}$ y $|D|$ es el número total de documentos.

La estimación de $\Pr(d \mid y, l')$ es más complicada. $\Pr(d \mid y, l')$ es la probabilidad de que un documento d pertenezca a la clase y dado que se consideran sólo documentos de longitud l'. Considerando que hay un elevado número de documentos diferentes, es muy difícil conseguir un número suficiente de ejemplos de entrenamiento para estimar la probabilidad sin conocimiento previo o sin hacer ciertas asunciones. En este caso la estimación es posible por la forma en que asumimos que se generan los documentos. Estamos suponiendo que la aparición de una palabra en un documento sólo depende de la clase del documento, que aparece independientemente del resto de palabras del documento y que no depende de la longitud del documento. Por tanto $\Pr(d \mid y, l')$ puede escribirse como:

$$\Pr(d \mid y, l') \approx \prod_{i=1}^{|d|} \Pr(\omega_i \mid y)$$

ω_i toma los valores de la secuencia de palabras del documento d que son consideradas como *features* (características). $|d|$ es el número de palabras del documento d. La estimación de $\Pr(d \mid y)$ se reduce a estimar cada $\Pr(\omega_i \mid y)$ de forma independiente. Una estimación Bayesiana para $\Pr(\omega_i \mid y)$ es la siguiente:

$$\widehat{\Pr}(\omega_i \mid y) = \frac{1 + TF(\omega_i, y)}{|F| + \sum_{\omega' \in |F|} TF(\omega', y)}$$

$TF(\omega, y)$ es el número total de veces que aparece la palabra ω en la clase de documentos y. A este estimador se lo conoce comúnmente como "Estimador de Laplace". Asume que la probabilidad de aparición de cada palabra es a priori la misma.

El resultado para la regla de clasificación combinando las ecuaciones anteriores es el siguiente:

$$h_{BAYES}(d) = \arg\max_{y\in\{-1,+1\}} \frac{\Pr(y)\cdot\prod_{i=1}^{|d|}\Pr(\omega_i, y)}{\sum_{y'\in\{-1,+1\}}\Pr(y')\cdot\prod_{i=1}^{|d|}\Pr(\omega_i, y')} =$$

$$= \arg\max_{y\in\{-1,+1\}} \frac{\Pr(y)\cdot\prod_{\omega\in X}\Pr(\omega\mid y)^{TF(\omega,d)}}{\sum_{y'\in\{-1,+1\}}\Pr(y')\cdot\prod_{\omega\in X}\Pr(\omega\mid y')^{TF(\omega,d)}}$$

Si no es necesario obtener el valor de $\Pr(y\mid d)$ como medida para la clasificación, el denominador podría eliminarse.

6 MÁQUINAS DE VECTORES SOPORTE

Las máquinas de vectores soporte fueron desarrolladas por Vapnik a partir del principio de *Minimización del Riesgo Estructural* de la teoría del aprendizaje estadístico.

La idea de este principio es encontrar una hipótesis *h* de un espacio de hipótesis *H* para la cual se garantiza la mínima probabilidad de error *Err(h)* para una muestra de entrenamiento dada *S* con *n* datos:

$$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \quad \vec{x}_i \in \Re^N, y_i \in \{-1, +1\}$$

La cota superior que se muestra a continuación conecta el error de la hipótesis *h* con el error $Err_{train}(h)$ de *h* en el conjunto de entrenamiento y la complejidad de *h*:

$$Err(h) \leq Err_{train}(h) + O\left(\frac{d \ln\left(\frac{n}{d}\right) - \ln(\eta)}{n}\right)$$

La cota se cumple con al menos una probabilidad de $1-\eta$. *d* representa la dimensión VC de Vapnik, que es una propiedad del espacio de hipótesis *H* e indica su expresividad. En la ecuación anterior se muestra el compromiso entre la complejidad del espacio de hipótesis y el error de entrenamiento. Este compromiso consiste en que un espacio de hipótesis con una dimensión baja no contendrá buenas funciones de aproximación y llevará a un error de entrenamiento alto; y por otro lado, un espacio de hipótesis de dimensión demasiado alta llevará a un error de entrenamiento pequeño pero el segundo término de la ecuación será muy grande. Este último caso se corresponde con el sobreentrenamiento de la máquina, que se ajusta de forma muy precisa a los datos de entrenamiento, pero que tiene un error muy alto al intentar clasificar nuevos datos.

Por tanto es fundamental elegir un espacio de hipótesis con la complejidad adecuada.

En el principio de Minimización del Riesgo Estructural, esto se consigue definiendo una estructura de espacios de hipótesis anidados H_i de forma que su dimensión d_i crece:

$$H_1 \subset H_2 \subset H_3 \subset \ldots \subset H_i \subset \ldots \quad y \quad \forall i : d_i \leq d_{i+1}$$

Esta estructura tiene que definirse a priori, antes incluso de analizar los datos de entrenamiento. El objetivo es encontrar el índice *i* que minimiza $Err(h)$.

Las máquinas de vectores soporte aprenden funciones de umbrales lineales como la siguiente:

$$h(\vec{x}) = sign\{\vec{\omega} \cdot \vec{x} + b\} = \begin{cases} +1, & si \quad \vec{\omega} \cdot \vec{x} + b > 0 \\ -1, & resto \end{cases}$$

Cada una de estas funciones corresponde a un hiperplano en el espacio de *features* (características). Esta función define que el lado del hiperplano donde se sitúa una muestra $\vec{x}$ determina cómo será clasificada por $h(\vec{x})$.

Estas funciones lineales tienen una dimensión de *N+1*, donde *N* es el número de características o *features*. Para problemas como el del presente proyecto, donde el número de *features* será muy alto (cientos o miles), la dimensión y por tanto la complejidad de las funciones sería demasiado alta.

A continuación se muestran algunas de las soluciones a este problema, con el fin de independizar la dimensión de las funciones del número de *features*.

Las soluciones no consisten, por tanto, en restringir el número de *features*, lo que haría inviable la clasificación, sino que se basan en el uso de una estructura refinada que no dependa de la dimensión del espacio de entrada.

Vapnik demostró que la distancia ∂ desde el hiperplano definido por la función hasta las muestras más cercanas puede ser usada para limitar la dimensión independientemente del número de *features*. Esta característica de las máquinas de vectores soporte es la que las hace interesantes para resolver la clasificación de textos.

A continuación se describirán a alto nivel las distintas aproximaciones al problema, deteniéndome más en la última, que será la solución utilizada finalmente, y que además es el resultado del trabajo de Emilio Parrado Hernández y su equipo del Departamento de Teoría de la Señal y Comunicaciones de la Universidad.

6.1 SVMs Lineales de Margen Duro

En esta solución, se asume que los datos de entrenamiento son separables por al menos un hiperplano *h'*. Esto significa que existen un vector de pesos $\vec{\omega}$ y un umbral *b'* que hacen que todos los ejemplos positivos de entrenamiento se sitúen a un lado del hiperplano, y todos los negativos al otro lado del mismo, o lo que es lo mismo, se asume el cumplimiento de:

$$y_i\left[\vec{\omega}'\cdot\vec{x}_i + b'\right] > 0$$

para cada muestra de entrenamiento $(\vec{x}_i, y_i)$. En general, pueden existir múltiples hiperplanos que separen los datos de entrenamiento sin error.

De todos ellos, la máquina de vectores soporte elige el de mayor margen ∂. Este margen es la distancia desde el hiperplano a las muestras de entrenamiento más cercanas al mismo. Para cada conjunto de datos de entrenamiento sólo existe un hiperplano con margen máximo. Las muestras más cercanas al hiperplano son las llamadas vectores soporte, y tienen una distancia de exactamente ∂ al hiperplano. El número de vectores soporte N_{SV} será el tamaño de la máquina.

La tarea de encontrar el hiperplano con margen máximo se resuelve con el siguiente problema de optimización, consistente en minimizar:

$$V(\vec{\omega}, b) = \frac{1}{2}\vec{\omega}\cdot\vec{\omega}, \ \forall_{i=1}^{n} : y_i\left[\vec{\omega}\cdot\vec{x}_i + b\right] \geq 1$$

Al hacer que la restricción sea ≥ 1, se está imponiendo la existencia de un cierto margen ∂ al hiperplano. De esto se puede deducir que $\partial = \frac{1}{\|\vec{\omega}\|}$,

donde $\|\vec{\omega}\|$ es la norma L_2 de $\vec{\omega}$. Por lo tanto, el vector de pesos $\vec{\omega}$ y el umbral *b* describen el hiperplano de margen máximo.

Según el Lema de Vapnik sobre dimensión de hiperplanos, considerando como hipótesis los hiperplanos $h(\vec{x}) = sign\{\vec{\omega} \cdot \vec{x} + b\} \geq 1$ en un espacion de N dimensiones, si todos los vectores de ejemplo $\vec{x}_i$ están contenidos en una bola de diámetro *R* y todos los ejemplos $\vec{x}_i$ cumplen que:

$$abs(\vec{\omega} \cdot \vec{x} + b) \geq 1$$

entonces el conjunto de hiperplanos tiene una dimensión *d* limitada por:

$$d \leq \min\left(\left[\frac{R^2}{\partial^2}\right], N\right) + 1$$

donde $\left[\frac{R^2}{\partial^2}\right]$ es la parte entera de $\frac{R^2}{\partial^2}$.

Por lo tanto, la dimensión es menor cuanto mayor es el margen, no depende del número de *features*, y depende de la longitud euclídea $\|\vec{\omega}\|$ del vector de pesos $\vec{\omega}$, optimizada por la máquina de vectores soporte.

Numéricamente, resolver el problema de optimización descrito anteriormente puede ser muy costoso. Sin embargo, el problema suele expresarse con su formulación dual, cuya solución es idéntica, y consiste en minimizar:

$$W(\vec{\alpha}) = -\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j)$$

sujeto a:

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

$$\forall i \in (1 \ldots n) : 0 \leq \alpha_i$$

Se conoce a la matriz *Q*, con $Q_{ij} = y_i y_j (\vec{x}_i \cdot \vec{x}_j)$ como la matriz Hessiana. El resultado del problema de optimización es un vector de

coeficientes $\vec{\alpha}^T = (\alpha_1, \ldots, \alpha_n)^T$, para los cuales $W(\vec{\alpha})$ es mínimo. Con estos coeficientes puede construirse el hiperplano:

$$\vec{\omega} \cdot \vec{x} = \left(\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \right) \cdot \vec{x} = \sum_{i=1}^{n} \alpha_i y_i (\vec{x}_i \cdot \vec{x})$$
$$b = y_{sv} - \vec{\omega} \cdot \vec{x}_{sv}$$

Con lo que el vector de pesos del hiperplano se construye como una combinación lineal de las muestras de entrenamiento. Sólo los vectores soporte tendrán un coeficiente α_i distinto de cero. Para calcular *b*, se puede utilizar un vector soporte cualquiera $\vec{x}_{sv}$ con su clase etiquetada y_{sv}.

6.2 SVMs de Margen Blando

Cuando los datos de entrenamiento no son separables por un hiperplano, se permiten durante el entrenamiento hasta un número determinado (parametrizable) de errores en la clasificación. A esta solución se le llama SVMs de Margen Blando.

De esta forma, la solución al problema pasará por minimizar simultáneamente este máximo de errores permitidos y la longitud del vector de pesos $\vec{\omega}$.

6.3 SVMs No Lineales

Cuando los datos no se ajustan a una solución lineal, es necesario utilizar una máquina de aprendizaje no lineal. Las Máquinas de Vectores Soporte son fácilmente transformables en máquinas no lineales mapeando los vectores $\vec{x}$ en un espacio de *features* de más dimensiones *X'*, utilizando un mapeo no lineal $\Phi(\vec{x}_i)$. La máquina aprendería la regla de clasificación lineal de margen máximo en el espacio *X'*, por lo que aunque la regla sea lineal en *X'*, es no lineal en el espacio de entrada original *X*. A continuación se muestra un ejemplo de mapeo no lineal:

$$\Phi\left((x_1, x_2)^T\right) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1\right)^T$$

Generalmente, el cálculo de estos "mapeos" es ineficiente. Sin embargo, se ha demostrado que para el entranamiento basta con poder

calcular los productos interiores $\Phi(\vec{x}_i)\cdot\Phi(\vec{x}_j)$. Estos productos pueden ser calculados muy eficientemente usando funciones *kernel* $K(\vec{x}_1,\vec{x}_2)$. Si una función $K(\vec{x}_1,\vec{x}_2)$ satisface la condición del Teorema de Mercer, es posible calcular el producto interior de los vectores $\vec{x}_1$ y $\vec{x}_2$ tras haber sido mapeados en un nuevo espacio de *features* por un mapeo no lineal Φ:

$$\Phi(\vec{x}_1)\cdot\Phi(\vec{x}_2)=K(\vec{x}_1,\vec{x}_2)$$

Dependiendo de la elección de la función *kernel*, las máquinas pueden aprender clasificadores polinómicos, RBF (Radial Basis Function), redes neuronales sigmoides de dos capas, o gaussianos:

$$K_{polinómico}(\vec{x}_1,\vec{x}_2)=(\vec{x}_1\cdot\vec{x}_2+1)^d$$

$$K_{rbf}(\vec{x}_1,\vec{x}_2)=\exp\left(-\gamma(\vec{x}_1-\vec{x}_2)^2\right)$$

$$K_{sigmoide}(\vec{x}_1,\vec{x}_2)=\tanh(s(\vec{x}_1\cdot\vec{x}_2)+c)$$

$$K_{gauss}(\vec{x}_1,\vec{x}_2)=\exp\left(-\frac{\|\vec{x}_1-\vec{x}_2\|^2}{2\sigma^2}\right)$$

El *kernel* para el ejemplo de mapeo anterior sería $K_{polinómico}(\vec{x}_1,\vec{x}_2)=(\vec{x}_1\cdot\vec{x}_2+1)^2$.

Calcular la función kernel es mucho más eficiente que enumerar todos los términos polinómicos como en la regresión polinomial. Para usar la función *kernel*, sólo es necesario sustituir en las ecuaciones cada ocurrencia del producto interior por la función *kernel* deseada.

6.4 SVMs Incrementales de Complejidad Controlada. GSVC:

Growing Support Vector Classifier

Como se ha explicado anteriormente los clasificadores basados en Máquinas de Vectores Soporte tradicionales buscan los umbrales entre las distintas clases o categorías que proporcionan el máximo margen, minimizando el riesgo estructural, dotando al clasificador de unas excelentes capacidades de generalización. Sin embargo, estos clasificadores

tienen algunas limitaciones que les hacen ser menos eficientes en algunos ámbitos.

Por ejemplo, son costosos computacionalmente por estar basados en programación cuadrática. Además, el tamaño del clasificador suele ser demasiado grande. Por otro lado, el proceso de entrenamiento no es accesible y está basado en un procedimiento de optimización determinístico calculado a priori con las muestras de entrenamiento.

Todo esto hace que sea difícil modificar el algoritmo para hacerlo más flexible. Por lo que sería muy conveniente poder monitorizar el comportamiento y la complejidad durante el entrenamiento.

Aunque se han hecho algunas aproximaciones para reducir los inconvenientes, no se había conseguido eliminarlos todos.

Con este tipo de clasificadores de vectores soporte incrementales (GSVC, del inglés Growing Support Vector Classifier) se consigue tener siempre bajo control el compromiso entre comportamiento y complejidad (o tamaño de la máquina). La naturaleza iterativa del GSVC permite la creación de métodos rápidos de clasificación sin coste adicional.

Sin embargo, el problema de seleccionar los hiperparámetros del kernel no está aún solucionado en GSVC, ya que por ejemplo, en el caso de un kernel Gaussiano, el valor óptimo de σ tiene que ser calculado antes del entrenamiento utilizando un proceso off-line.

GSVC facilita la introducción de aproximaciones multi-kernel, MK-GSVC. Para kernels Gaussianos, el método MK-GSVC comienza con un valor inicial de σ grande y progresivamente va disminuyendo su valor con los nuevos kernels añadidos, de forma que se obtienen soluciones más finas según crece la máquina.

Como se explicó en apartados anteriores, los vectores $\vec{x}_i$ con coeficientes $\alpha_i > 0$ son los llamados Vectores Soporte (VS), y el tamaño de la máquina es el número de VS, N_{SV}. En los problemas en los que las clases se solapan demasiado, se encuentran demasiados VS, lo que hace que las máquinas sean muy grandes. Para resolver esto, existen algoritmos alternativos que consiguen aproximarse mucho a la solución con una

máquina más compacta, como es el Weighted Least Squares Support Vector Classifier (WLS-SVC). Este algoritmo construye una aproximación semiparamétrica a partir de una arquitectura estimada de *R* nodos. La formulación es la siguiente:

$$\hat{h}(\vec{x}) = sign\left(\sum_{k=1}^{R} \beta_k \vec{\phi}(\vec{c}_k)\vec{\phi}(\vec{x}) + b\right) = sign\left(\sum_{k=1}^{R} \beta_k k(\vec{c}_k, \vec{x}) + b\right)$$

Donde $\{\vec{c}_k\}$ son los *R* centroides correspondientes a las principales direcciones del subespacio *H*. Los coeficientes $\{\beta_k\}$ son inicializados aleatoriamente y resueltos iterativamente hasta su convergencia.

Con todo esto, GSVC comienza con una máquina reducida, con unos pocos centroides seleccionados aleatoriamente, y en cada paso del algoritmo, se añaden nuevos centroides para mejorar la clasificación. Cada vez que la arquitectura crece, WLS-SVC permite actualizar la solución, sin la necesidad de resolverlo desde cero.

GSVC será el algoritmo utilizado en este proyecto para clasificar correos electrónicos en spam y ham. En apartados posteriores se detallará cómo se ha integrado el módulo de GSVC en el software desarrollado, y se explicará cuáles son sus entradas y sus salidas.

7 DESARROLLO DEL CLASIFICADOR

En este capítulo se describirá el proceso de desarrollo del detector de spam, o clasificador de correos electrónicos, desde la selección de *features* o características hasta su integración con MailScanner, y las pruebas de integración correspondientes.

Se mostrarán además los experimentos realizados, sus resultados y las conclusiones extraídas de cada uno de ellos.

Se irá haciendo referencia a cada uno de los anexos que contienen el código fuente desarrollado.

Como lenguaje de programación para la implementación del clasificador se eligió Perl, por las siguientes razones fundamentales:

- El grueso del procesado que llevará el clasificador será procesado de textos, y Perl es un lenguaje muy potente en este ámbito.
- El software de MailScanner está desarrollado en Perl, con lo que la integración será mucho más sencilla.

No obstante, se harán llamadas a programas externos, como las rutinas de entrenamiento y de clasificación de GSVC, desarrolladas en lenguaje C.

7.1 DESCRIPCIÓN DE LA PLATAFORMA DE DESARROLLO

El desarrollo del clasificador de Correos se llevó a cabo sobre una plataforma Linux con las siguientes características:

- Fedora Core Linux 6 instalado sobre partición de disco de 60 GBytes.
- Procesador Intel Pentium D 2,8 GHz
- Memoria RAM: 1 GByte

Fue necesario instalar las siguientes librerías adicionales a las de Perl:

- Mail::MboxParser – Proporciona herramientas para lectura de buzones de correo en formato mbox.

- HTML::Tagset – Proporciona un conjunto de etiquetas html que como se explicará más adelante serán usadas como stopwords.

7.2 SELECCIÓN DE *FEATURES*

El primer problema que debe ser resuelto es la selección de las características o *features* que serán utilizadas para representar cada uno de los correos electrónicos como vectores, y que serán utilizadas posteriormente en las tareas de entrenamiento y clasificación.

Como se indicó en apartados anteriores, la selección de *features* depende en primer lugar del nivel elegido para la representación de textos.

En este proyecto, se ha elegido el nivel "Palabra" para representar los correos electrónicos por diferentes motivos:

- Es el que menos complejidad de desarrollo requiere, ya que no es necesario analizar las raíces morfológicas de las palabras, ni hacer subdivisiones, ni componer palabras nuevas. Sólo se tendrán en cuenta las palabras que compongan el cuerpo del correo electrónico.

- Es independiente del idioma, o más concretamente, pueden analizarse correos electrónicos escritos en diferentes idiomas, sin que el clasificador deba conocer ninguna particularidad de cada idioma. Dos palabras con el mismo significado en dos idiomas diferentes, serán tratadas como dos palabras distintas.

Por otro lado, el objetivo es clasificar correos electrónicos por su contenido. Como se indicará en apartados posteriores, MailScanner junto con SpamAssassin ya disponen de reglas de clasificación, listas blancas y negras, e incluso un clasificador bayesiano para detectar spam. Estas dos herramientas ya tienen muy desarrollada la detección de spam por las cabeceras de los correos, listas de IPs, detección de phising, análisis de asuntos, etc. Por tanto, el clasificador que se desarrollará en este proyecto analizará única y exclusivamente el texto contenido dentro del cuerpo de los correos electrónicos.

Como ya se explicó, al nivel "Palabra" de representación de textos, las dos medidas más comunes de medición de la relevancia de una palabra en un texto son el Term Frequency (TF) y el Term Frequency Inverse Document Frequency (TFIDF).

A priori, parece lógico pensar que el uso del TFIDF proporcionará mejores resultados que el TF, ya que es una medida que tiene en cuenta las ocurrencias de una palabra no sólo dentro del documento concreto, sino en todo el conjunto de documentos de análisis. Sin embargo, la cantidad de procesado necesaria si se usa TFIDF será mucho mayor. Por ello, uno de los objetivos de este proyecto es comparar ambas representaciones en cuanto a los resultados proporcionados por el clasificador de textos y al tiempo de procesado requerido por cada uno de ellos.

La selección de *features* se llevará a cabo mediante el análisis exhaustivo de los datos de entrenamiento, recorriendo los buzones que contienen los correos de entrenamiento, y calculando el TFIDF para todas las palabras encontradas. Este procesado es costoso, dependiendo del procesador y de la memoria de la máquina donde se ejecute, puede llegar a tardar algunas horas. Sin embargo sólo es necesario ejecutarlo una vez para seleccionar las *features*.

En un entorno real, sería recomendable ejecutarlo con una periodicidad mensual para actualizar las *features*.

En el Anexo B se muestra el código que lee los correos electrónicos y calcula el TFIDF para todas las palabras de los buzones de entrenamiento y obtiene el listado de *features*.

Como puede observarse, y tal y como se indicó en el apartado correspondiente, para seleccionar las *features* se eliminaron aquellas palabras que no son relevantes en la clasificación, las llamadas "stopwords".

Dado que los correos electrónicos pueden estar escritos en varios idiomas, se emplearon stopwords de los siguientes idiomas: Inglés, Español, Catalán, Checo, Danés, Holandés, Francés, Alemán, Húngaro, Italiano, Noruego, Polaco, Portugués y Turco.

Una característica de los correos electrónicos cada vez más común es que estén escritos en html. Por ello, se hicieron pruebas incluyendo y sin incluir un listado de etiquetas html en el listado de stopwords.

Finalmente, se comprobó que los resultados eran mejores si se incluían las etiquetas html en el listado de stopwords, no pudiendo ser por tanto potenciales *features*.

Por último, se ordenaron las palabras de mayor a menor valor de TFIDF, obteniendo como salida varios listados de *features* en función del número total seleccionado.

Otro preprocesado llevado a cabo sobre el texto fue la eliminación de tildes y diéresis. El beneficio de este tratamiento fue por un lado la reducción del número de palabras distintas existentes en los correos electrónicos y por otro evitar el ruido provocado por errores ortográficos, por otro lado muy comunes. Se asumen los errores que este tratamiento puede provocar al tener en cuenta dos palabras distintas como si fuera una única, ya que el impacto será menor que el de tratarlas por separado.

Por las mismas razones, se convirtieron en todos los procesos las mayúsculas a minúsculas.

A cada *feature* se le asignará una clave, numérico autoincremental según orden de mayor a menor, de forma que el *feature* con mayor TFIDF tendrá asignado la clave 1, y el *feature* con menor TFIDF tendrá asignado la clave N, donde N es el número de *features* total.

Este listado se almacenará en un fichero para poder ser utilizado por los procesos de entrenamiento y de clasificación.

7.3 ENTRENAMIENTO

Una vez extraídas las *features* que compondrán el espacio de entrada para el entrenamiento y la clasificación, comenzará el proceso de entrenamiento del clasificador. El primer paso será preparar juegos de datos de entrenamiento y de test. Deberán ser representados en el espacio de *features* seleccionadas y posteriormente etiquetados para indicar a los procesos de train y de test a qué clase pertenece cada correo.

7.3.1 Preparación de datos de Entrenamiento y Validación

Será necesario generar dos conjuntos de correos etiquetados:

- Conjunto de entrenamiento (train): Será el conjunto de correos utilizados por el proceso de entrenamiento para entrenar la máquina y obtener la regla de validación con parámetros concretos. A su vez, del conjunto de entrenamiento se extraerá un conjunto de correos que será utilizado para validación. Se ajustará el clasificador para optimizar los resultados obtenidos al clasificar este conjunto de validación.

- Conjunto de test: Será el conjunto de correos que permitirán comprobar la capacidad de generalización de la máquina entrenada. Será un conjunto independiente del de train, y será presentado una sola vez al clasificador para obtener resultados de la capacidad de generalización del mismo.

Para generar estos grupos de correos se tomaron los buzones de correo disponibles, y se dividieron en primer lugar en dos grupos: spam y ham.

De todos los correos spam se seleccionaron aleatoriamente el 40% de los mismos para entrenamiento, y el 60% restante para test. El mismo procedimiento se llevó a cabo con los correos ham.

- Conjunto train: 40% de los correos spam + 40% de los correos ham

- Conjunto test: 60% de los correos spam + 60 % de los correos ham

Como salida, el subproceso deberá componer un fichero de texto que sirva de entrada al entrenamiento de GSVC, que contendrá el 40% del total de correos disponibles. Este fichero se dividirá a su vez en el fichero de entrenamiento y el de validación, con proporción 40% - 60% respectivamente. El fichero contendrá una línea de texto por cada correo, y cuyo formato será el siguiente:

$Y1\_F1{:}TFIDF_{11}\_F2{:}TFIDF_{21}\_F3{:}TFIDF_{31}\_\ldots\ldots\ldots\ldots FN{:}TFIDF_{N1}$

$Y2\_F1{:}TFIDF_{12}\_F2{:}TFIDF_{22}\_F3{:}TFIDF_{32}\_\ldots\ldots\ldots\ldots FN{:}TFIDF_{N2}$

…

YM_F1:TFIDF$_{1M}$_F2:TFIDF$_{2M}$_F3:TFIDF$_{3M}$_……FN:TFIDF$_{NM}$

donde Y1 hasta YM son las etiquetas de los correos 1 hasta M, y en el caso que nos ocupa tendrán dos valores posibles, -1 para etiquetar un correo ham y +1 para etiquetar un correo spam. La etiqueta serán los primeros caracteres de cada línea y estarán separados de la primera clave de *feature* por un espacio, representado en la notación anterior con un "_".

F1 hasta FN son las claves de los *features* seleccionados previamente y TFIDF$_{1M}$ hasta TFIDF$_{NM}$ son los valores de TFIDF de cada *feature* en el correo M. Entre un valor de TFIDF y la siguiente clave, debe indicarse un espacio en blanco, que se ha indicado en la notación anterior con un "_".

Cuando el TFIDF para un determinado *feature* sea igual a 0, éste no se incluirá en el fichero. Con el ejemplo de notación anterior, si TFIDF$_{21}$ fuera igual a 0, la línea se compondría como se muestra a continuación:

Y1_F1:TFIDF$_{11}$_F3:TFIDF$_{31}$_……FN:TFIDF$_{N1}$

Se generará un segundo fichero para test, que contendrá el 60% del total de correos disponibles. Su formato será idéntico al fichero de entrenamiento.

Se muestra en el Anexo C el código fuente que genera los ficheros de train y test.

7.3.2 ENTRENAMIENTO GSVC

El proceso de entrenamiento de GSVC recibirá el fichero de train descrito en el apartado anterior. El motor de entrenamiento leerá el fichero, con los datos etiquetados y obtendrá de forma iterativa, como se describió en el apartado 6.4, la regla de clasificación, optimizando el número de centroides del kernel Gaussiano. Este proceso proporcionará como salidas los dos ficheros siguientes:

- modelo.txt
- centros_matlab.txt

7.3.3 VALIDACIÓN GSVC

El proceso de validación de GSVC recibirá el fichero de validación descrito en el apartado anterior. Para ejecutar la validación de la máquina se utilizará el módulo de clasificación de GSVC, que cuando recibe ficheros de datos etiquetados, entiende que se trata de una validación, clasifica los datos, y tras comparar los resultados con las etiquetas proporcionadas, obtiene un valor porcentual que representa el acierto en la clasificación.

Este proceso devuelve también a la salida dos ficheros.

- `output.txt`: donde se indicará la clasificación del correo, a través de un valor numérico para cada uno de ellos. Este fichero permitirá realizar análisis más detallados del resultado del clasificador.

- `gmon.out`: fichero con datos de monitorización del proceso, que dará información de rendimiento del proceso. Será necesario usar la utilidad de UNIX `gprof` para leerlo.

El valor de σ será optimizado iterativamente mediante sucesivas ejecuciones de entrenamientos y validaciones.

7.4 Clasificación (Test)

El proceso de clasificación consta de dos subprocesos: la representación del correo electrónico en base a los *features* seleccionadas previamente, y la clasificación propiamente dicha con GSVC.

7.4.1 Representación de Correos Electrónicos

El subproceso de representación de correos en base a los *features* seleccionadas previamente necesita de los siguientes datos de entrada:

- El correo a analizar.

- El listado de *features*, con sus claves asociadas.

- Frecuencia de Documento (Document Frecuency) de cada *feature*. Será una tabla hash cuyas claves serán los *features* de base, y cuyos valores asociados representarán el número de

documentos que contienen cada *feature* hasta ese momento. Este dato se actualizará con el análisis de cada nuevo correo analizado.

El correo será recibido desde MailScanner, como un objeto mensaje. A partir de ahí, será necesario obtener el cuerpo del correo y analizarlo para generar su representación.

Como salida, el subproceso deberá componer un fichero de texto que sirva de entrada al clasificador de GSVC. El fichero contendrá una única línea de texto que representará al correo, y cuyo formato será el siguiente:

$F1:TFIDF_1\_F2:TFIDF_2\_F3:TFIDF_3\_\ldots\ldots\ldots\ldots FN:TFIDF_N$

donde F1 hasta FN son las claves de los *features* seleccionados previamente y $TFIDF_1$ hasta $TFIDF_N$ son los valores de TFIDF de cada *feature* en el correo. Entre un valor de TFIDF y la siguiente clave, debe indicarse un espacio en blanco, que se ha indicado en la notación anterior con un "_".

Cuando el TFIDF para un determinado *feature* sea igual a 0, éste no se incluirá en el fichero. Con el ejemplo de notación anterior, si $TFIDF_2$ fuera igual a 0, la línea se compondría como se muestra a continuación:

$F1:TFIDF_1\_F3:TFIDF_3\_\ldots\ldots\ldots\ldots FN:TFIDF_N$

Una vez compuesto el fichero, éste se pasará como entrada al clasificador de GSVC.

7.4.2 Clasificación Del Correo

El siguiente subproceso será el de clasificación propiamente dicho. El motor de clasificación GSVC recibirá como entradas al proceso los siguientes ficheros:

- Fichero que representa al correo electrónico en la base de *features* seleccionados. El formato del fichero será el indicado en el apartado anterior.
- Ficheros de salida del proceso de entrenamiento y que definen al clasificador. Entre otros datos, muestran el número de centroides al finalizar el entrenamiento y el valor de σ del kernel Gaussiano:
  - `modelo.txt`

- `centros_matlab.txt`

Como salidas del subproceso de clasificación se obtendrán los ficheros:

- `output.txt`: donde se indicará la clasificación del correo, a través de un valor numérico. Este valor numérico podrá ser utilizado para informar a MailScanner un indicador de la certeza de si el correo es o no spam (scoring).

- `gmon.out`: fichero con datos de monitorización del proceso, que dará información de rendimiento del proceso. Será necesario usar la utilidad de UNIX `gprof` para leerlo.

Se muestra en el Anexo D el código fuente del módulo que recibe el correo de MailScanner, lleva a cabo su representación en la base de *features* seleccionadas, ejecuta la clasificación de GSVC y devuelve el scoring a MailScanner.

7.5 Resultados Experimentales

Para comprobar el comportamiento del software desarrollado, se realizaron numerosos casos de prueba. Se detallan a continuación algunos de ellos.

7.5.1 Experimento 1

En este caso de prueba, se seleccionaron un buzón de correo de spam y otro de ham para ejecutar todos los subprocesos:

- Extracción de *features*

- Entrenamiento

- Validación

A continuación se muestra el tamaño de los buzones en nº de correos electrónicos:

Buzón	**Nº de correos**
Spam	2295

Ham	2110

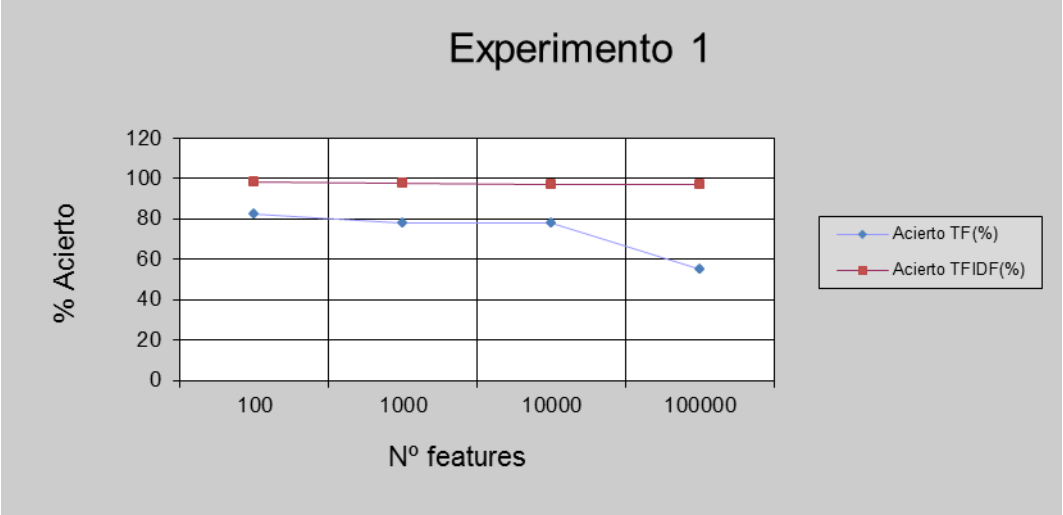
Se ejecutaron el entrenamiento y la clasificación ocho veces, variando el número de *features* y el tipo de medida de pesos, usando TF y TFIDF.

Como siempre, se utilizó el 40% de los correos de cada buzón para el entrenamiento (train + validación) y el 60% de los correos de cada buzón para test.

Se muestran en una tabla a continuación los resultados del test:

Nº *features*	TF		TFIDF	
	%Acierto TF	NºCentros	%Acierto TFIDF	NºCentros
100	82,643428	138	98,257081	66
1000	77,850399	66	97,603486	74
10000	77,850399	138	97,385621	114
100000	55,192447	82	96,949891	114

Y su representación gráfica:



Del resultado de esta ejecución, se pueden extraer las siguientes conclusiones:

- El uso de TFIDF como medida de pesos para los vectores que representan a los correos electrónicos proporciona mejores prestaciones en cuanto a clasificación que cuando se usa el TF.

- Al aumentar el número de *features* que forman la base para representación de los correos, el acierto en la clasificación disminuye, lo que denota un posible sobreentrenamiento. Éste no es el resultado esperado, y se debe muy probablemente a que es conjunto de datos de entrenamiento es demasiado reducido, por lo que la máquina no llega a tener la capacidad de generalizar tras el entrenamiento.

7.5.2 EXPERIMENTO 2

En este caso de prueba, se utilizaron todos los buzones de correo spam y todos los de ham para ejecutar la extracción de *features*, el entrenamiento y el test.

A continuación se muestra el tamaño de los buzones en nº de correos electrónicos:

Buzón	**Nº de correos**
Spam	2554
Ham	6215

Se ejecutaron el entrenamiento y la clasificación ocho veces, variando el número de *features* y el tipo de medida de pesos, usando TF y TFIDF.

Se utilizó el 40% de los correos de cada tipo para el entrenamiento (train + validación), y el 60% de los correos de cada tipo para el test.

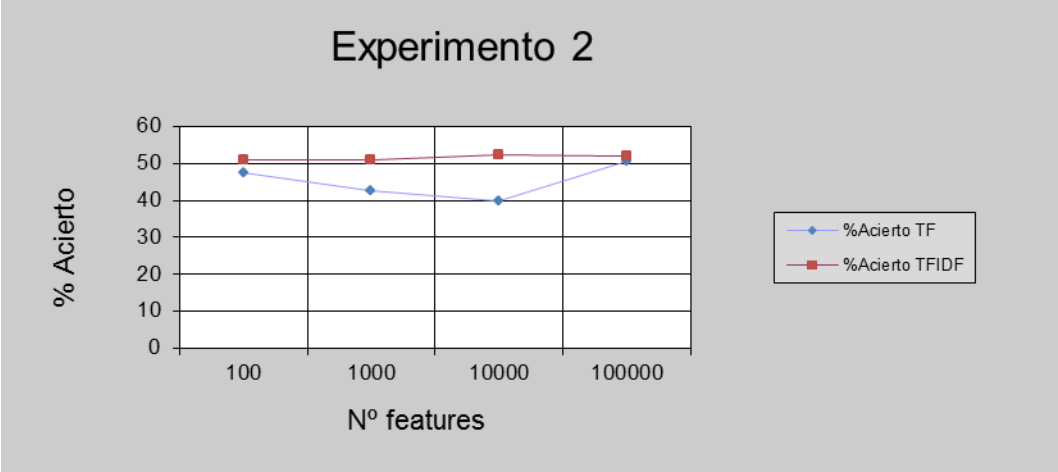
Se muestran en una tabla a continuación los resultados del entrenamiento y del test:

RESULTADO ENTRENAMIENTO GSVC					
Tipo Análisis	**Nº *features***	**Nº Iteraciones**	**Nº V. S.**	**Nº Cent.**	**CE Validacion**
TF	100	13	1386	50	2,09E+00
TF	1000	15	2986	114	9,60E+00
TF	10000	21	3413	138	1,07E+01
TF	100000	14	3399	58	9,30E+00
TFIDF	100	13	490	50	1,44E+00
TFIDF	1000	31	182	122	4,02E-01
TFIDF	10000	13	172	82	3,02E-01

TFIDF	100000	15	193	90	7,04E-01

RESULTADO TEST		
Nº *features*	%Acierto TF	%Acierto TFIDF
100	47,624477	51,064234
1000	42,474344	50,893197
10000	39,870772	52,280502
100000	50,494109	52,166477

A continuación se muestran los resultados del test de forma gráfica:



En este experimento, la diferencia con respecto al anterior fue el aumento del número de correos electrónicos en el tratamiento, aumentando el número de correos para entrenamiento y para test. Se pueden extraer las siguientes conclusiones:

- Al aumentar el conjunto de datos de entrenamiento, la máquina pierde la capacidad de generalizar, y el porcentaje de acierto se asemeja al que produciría un decisor aleatorio, es decir, el peor de los casos.

- Dado que estos no son en absoluto los resultados esperados, se realizó un análisis del proceso, y se detectó un posible foco de problemas. Los datos de entrenamiento se estaban pasando ordenados a la máquina, incluyendo todos los correos ham al principio y todos los correos spam al final. Esto puede hacer que la máquina, al ser iterativa, vaya modificando la regla de

clasificación siempre en la misma dirección durante muchas iteraciones, y que finalmente no sea capaz de adaptarse.

7.5.3 EXPERIMENTO 3

En este experimento se tomaron exactamente los mismos datos para ejecutar los procesos de extracción de *features*, de entrenamiento y de test que en el experimento 2.

La única diferencia, y crítica, como veremos a continuación fue que para el proceso de entrenamiento se "aleatorizaron" los datos de entrada.

En el experimento 2 los datos de entrada al entrenamiento se generaron leyendo y representando primero todos los correos ham y posteriormente todos los correos spam. De esta forma en el fichero de train las 6215 primeras líneas correspondían a correos ham y las 2554 siguientes a correos spam.

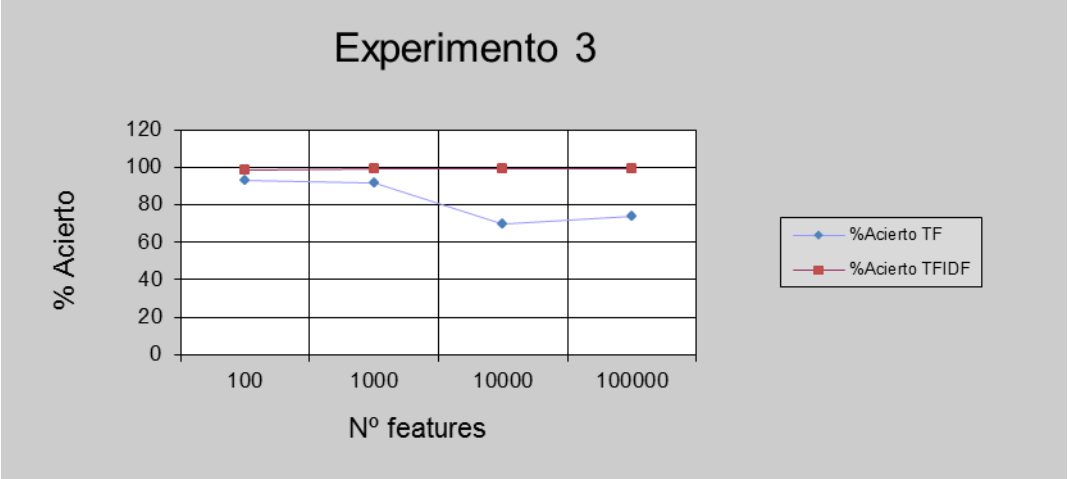
En este experimento, se fraccionó el fichero de entrenamiento (train + validación) en trozos de 50 líneas (50 correos), y se mezclaron aleatoriamente para componer un nuevo fichero de entrenamiento "aleatorizado".

Se muestran en una tabla a continuación los resultados del entrenamiento y de la validación:

RESULTADO ENTRENAMIENTO GSVC					
Tipo Análisis	**Nº *features***	**Nº Iteraciones**	**Nº V. S.**	**Nº Cent.**	**CE Validacion**
TF	100	24	1792	187	5.680120e+00
TF	1000	17	3393	106	7.814629e+00
TF	10000	9	3426	66	4.460227e+01
TF	100000	17	3442	122	1.684712e+01
TFIDF	100	14	565	82	2.785027e+00
TFIDF	1000	25	148	146	6.962567e-01
TFIDF	10000	37	161	202	6.499392e-01
TFIDF	100000	20	214	122	2.475248e-01

RESULTADO TEST		
Nº *features*	**%Acierto TF**	**%Acierto TFIDF**
100	92,769026	98,253938
1000	91,364585	98,823306
10000	69,519833	99,383906
100000	73,771114	99,502885

A continuación se muestran los resultados del test de forma gráfica:



Como puede observarse los resultados mejoran radicalmente simplemente con la tarea de "aleatorización" del orden de los datos de entrada. Las conclusiones son las siguientes:

- El entrenamiento de la máquina no es independiente del orden de los datos de entrenamiento. Si la máquina recibe primero todos los pertenecientes a una clase y al final todos los pertenecientes a la otra, el entrenamiento no es capaz de generar una regla de clasificación que generalice bien.

- Se confirma que el uso de TFIDF como medida de peso para los vectores que representan a los correos electrónicos proporciona resultados mucho mejores que el uso del TF. Como ya se ha comentado en otros apartados, esto se debe a que el TFIDF es capaz de tener en cuenta aquellas palabras que son muy frecuentes en todos los correos (documentos) para que no sean representativas ni en el entrenamiento ni en el test.

- Con el uso de TF la máquina empieza a estar sobreentrenada cuando se usan más de unos cientos de *features*. Sin embargo, con el uso de TFIDF la máquina parece no llegar al sobreentrenamiento.

7.5.4 EXPERIMENTO 4

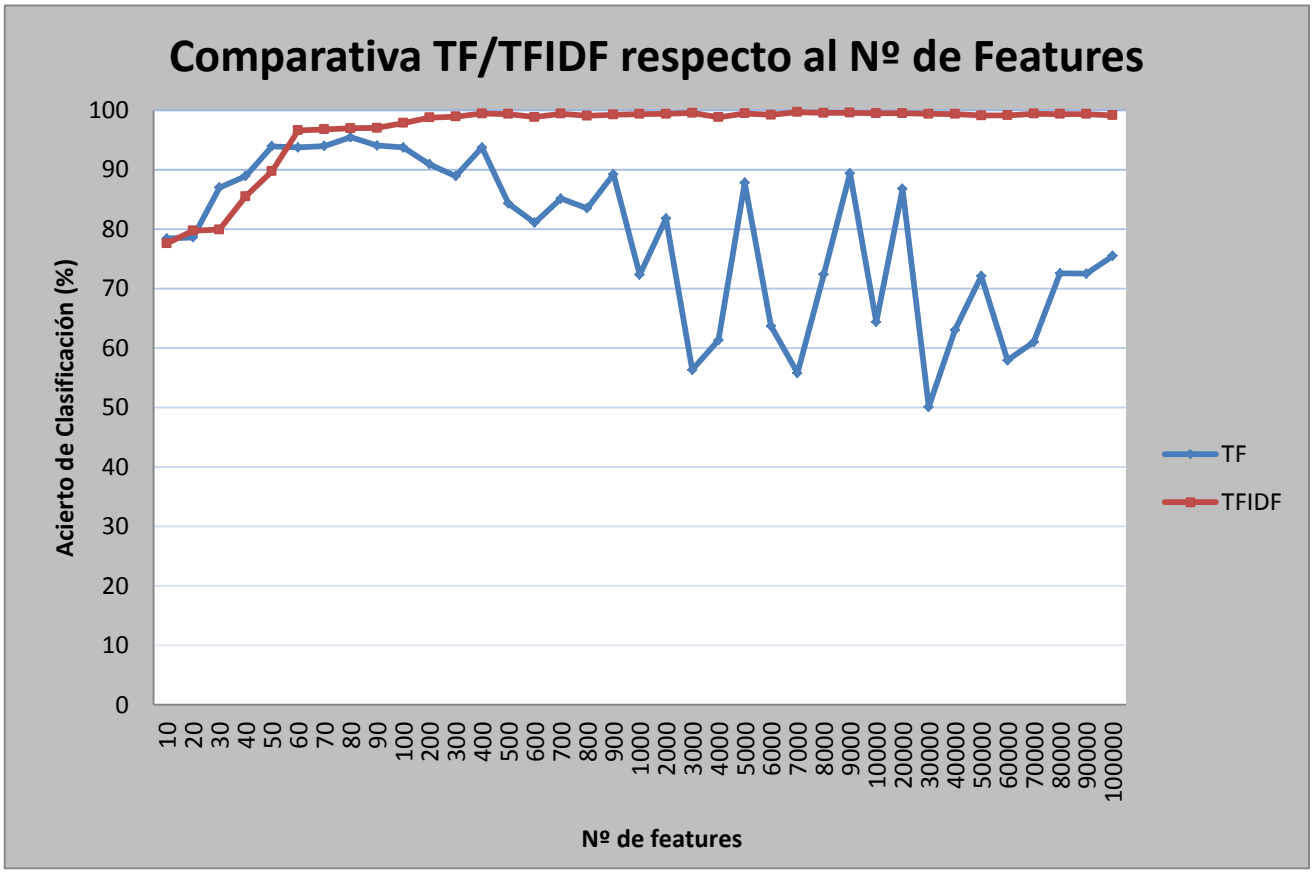
En este experimento se siguió el procedimiento del experimento 3, utilizando todos los buzones disponibles, y "aleatorizando" el fichero de entrenamiento. Además, se amplió el número de ejecuciones para observar la evolución del acierto de clasificación con el número de *features* y como siempre comparando los resultados usando TF y TFIDF.

A continuación se muestran los resultados globales del algoritmo, obteniendo el porcentaje de acierto en la clasificación con dos variables: el número de *features* seleccionadas desde un mínimo de 10 hasta el máximo de 100000, distinguiendo si el análisis se ejecutó con Term Frequency (TF) o con Term Frequency Inverse Document Frequency (TFIDF) como medida para los vectores de pesos.

Nº *Features*	% acierto (TF)	% acierto (TFIDF)
10	78,414097	77,593112
20	78,614337	79,735683
30	86,984381	79,915899
40	88,926712	85,522627
50	93,932719	89,727673
60	93,732479	96,615939
70	93,992791	96,776131
80	95,454545	96,956348
90	94,072887	97,01642
100	93,752503	97,857429
200	90,929115	98,778534
300	88,926712	98,918702
400	93,752503	99,439327
500	84,301161	99,359231
600	81,097317	98,838606
700	85,142171	99,419303
800	83,5002	99,078895
900	89,227072	99,259111
1000	72,326792	99,359231
2000	81,778134	99,399279
3000	56,327593	99,559471
4000	61,3336	98,818582
5000	87,805366	99,479375
6000	63,696436	99,239087
7000	55,786944	99,679616
8000	72,36684	99,559471
9000	89,387265	99,579495

10000	64,357229	99,499399
20000	86,784141	99,499399
30000	50,080096	99,399279
40000	63,055667	99,359231
50000	72,106528	99,118943
60000	57,949539	99,158991
70000	60,993192	99,419303
80000	72,56708	99,359231
90000	72,507008	99,359231
100000	75,490589	99,158991

Se muestran a continuación los resultados de forma gráfica.



De estos resultados pueden extraerse varias conclusiones:

- Como cabía esperar, el uso de TFIDF proporciona índices de generalización sensiblemente mayores que con el uso de TF.

- Utilizando TF para la clasificación, la máquina llega al sobreentrenamiento cuando el número de *features* es todavía bajo. El porcentaje de acierto llega a estar en niveles buenos (>95%) con el uso de un número de *features* cercano a 100. Sin embargo, cuando aumenta el número de *features* en el análisis la clasificación sufre un ruido excesivo, haciendo que los resultados sean casi aleatorios.

- La máquina no llega al sobreentrenamiento aunque se aumente mucho el número de *features* cuando se usa el TFIDF como medida de peso. Esto es debido a que los pesos medidos en TFIDF de las palabras muy frecuentes en todos los documentos son muy bajos y no llegan a tener efecto sobre la regla de clasificación.

- Por lo anterior, los resultados del clasificador TFIDF según aumenta el número de *features* convergen en un porcentaje de acierto de alrededor del 99,5% a partir de 3000 *features*.

7.5.5 EXPERIMENTO 5

En este experimento se extrajeron los parámetros “Falso positivo” y “Falso negativo”.

Como se dijo en apartados anteriores, el parámetro más perjudicial en la clasificación de correos electrónicos es el “Falso positivo”, es decir, clasificar como spam correos que no lo son, puesto que supone la potencial pérdida de correos deseados, y eventualmente importantes.

Para realizar el cálculo, se analizaron los resultados del clasificador para una de las ejecuciones del experimento anterior. En concreto, se tomaron los resultados para el análisis basado en TFIDF y 10000 *features*.

El clasificador GSVC devuelve un fichero con los resultados de la clasificación de todos los datos de entrada, devolviendo un número real por cada dato de entrada. Los valores de las etiquetas durante todos los experimentos fueron -1 para ham y +1 para spam. Los números devueltos por el clasificador son negativos y positivos, de forma que deben

interpretarse como ham y spam respectivamente, siendo 0 el umbral de decisión.

Los resultados obtenidos para dichos parámetros fueron los siguientes:

Parámetro	Valor
Acierto clasificación	99,499399 %
Falso positivo	0,6452837 %
Falso negativo	0,1708104 %

Aunque los parámetros son muy buenos (sólo seis de cada mil correos son clasificados como spam erróneamente, y sólo 2 de cada mil correos son clasificados como ham erróneamente), se comprobó que es posible reducir los falsos positivos modificando levemente el umbral de decisión, ya que para la gran mayoría de los errores de clasificación, el clasificador había devuelto un valor muy cercano a 0, mientras que los aciertos eran valores fuertemente positivos o fuertemente negativos (valores de varias unidades).

Estableciendo el umbral de decisión en 0,5, los resultados fueron los siguientes:

Parámetro	Valor
Acierto clasificación	99,37003226 %
Falso positivo	0,284684001%
Falso negativo	0,645283735 %

Con esta modificación del umbral, el acierto de clasificación general bajó algo más de una décima porcentual, mientras que efectivamente el "Falso positivo" disminuyó a alrededor de 3 correos electrónicos de cada

1000 clasificados como spam sin serlo, a cambio de aumentar el "Falso negativo" de 2 a 6 correos de cada 1000.

Se hizo la prueba de aumentar todavía más el umbral hasta llegar a un valor de 1, para intentar reducir el "Falso positivo" todavía más, y los resultados fueron los siguientes:

Parámetro	Valor
Acierto clasificación	97,71664452%
Falso positivo	0,018978933 %
Falso negativo	3,264376542 %

En este caso, el falso positivo se redujo a sólo 2 correos de cada 10000 clasificados como spam sin serlo. A cambio, el falso negativo aumentó hasta alrededor de 3 correos de cada 100 clasificados como ham, siendo realmente spam.

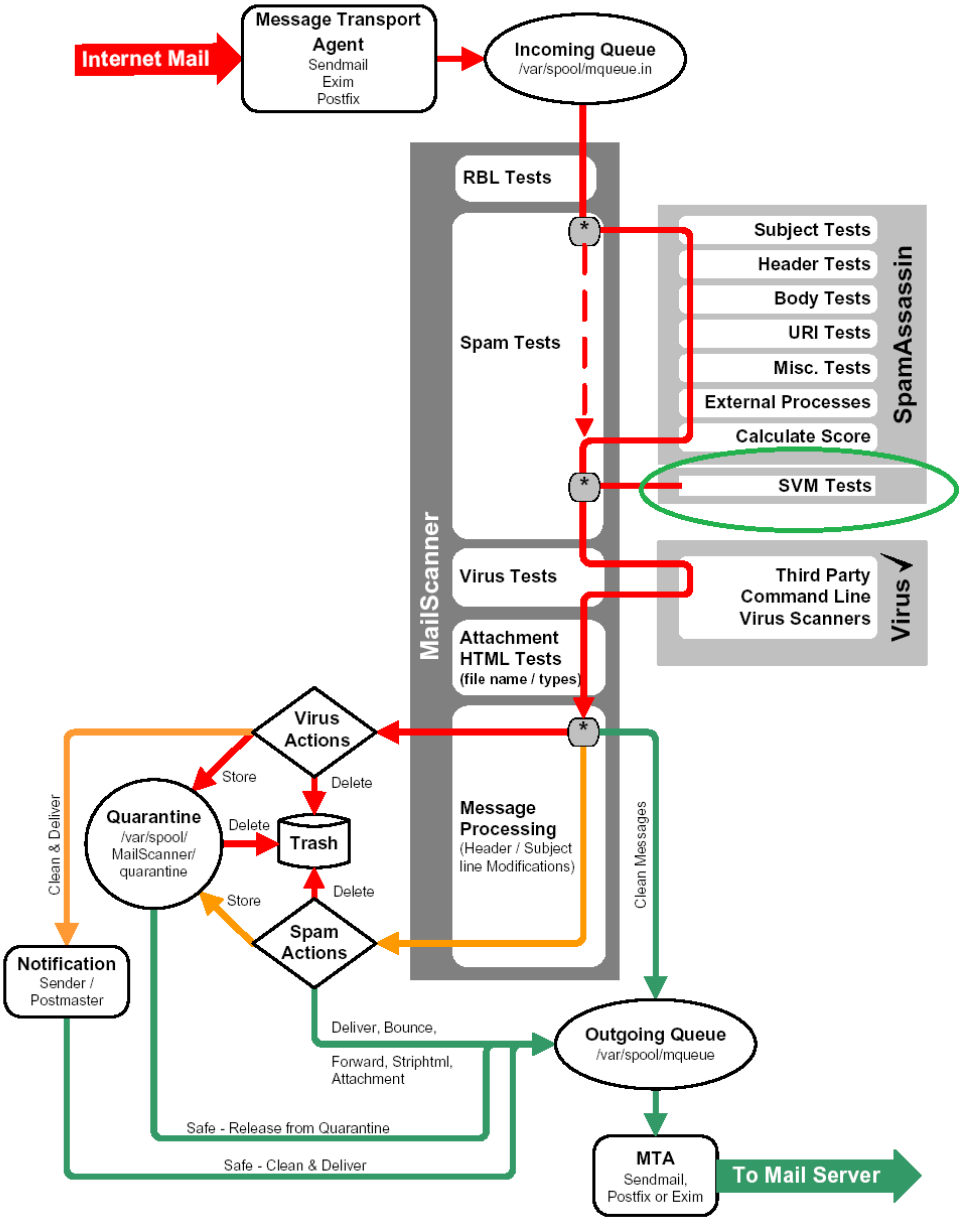
Con todo esto, se puede concluir que el clasificador tiene unas muy buenas prestaciones y que además, es fácilmente configurable para adaptarse a las necesidades de cada usuario.

Por otro lado, y con el objetivo en la integración del clasificador en MailScanner, se observa que el valor real devuelto por el clasificador GSVC podría ser devuelto a MailScanner, incluyéndolo en el asunto del correo a modo informativo, de manera que el usuario pudiera tener una idea de con cuánta probabilidad un correo es Spam o no.

7.6 Integración con MailScanner y SpamAssassin

Como se indicó al principio del documento, MailScanner será el sistema sobre el que se incorporará el clasificador de textos externo desarrollado. MailScanner utiliza actualmente la herramienta SpamAssassin como clasificador. El funcionamiento de estas dos herramientas se describió en apartados anteriores.

A continuación se muestra el diagrama de flujo de un correo electrónico en su paso a través de MailScanner, incluyendo el análisis desarrollado en este proyecto, y denotado en el diagrama como "SVM Tests":



Modificación propia a partir del original: **MailScanner, Open source Secure Mail Gateway Software. Administrators Guide, Version 1.0.1**

ILUSTRACIÓN 2 - FLUJO MAILSCANNER MODIFICADO

Durante el desarrollo de este proyecto, MailScanner sufrió cambios que afectan a la integración del módulo clasificador mediante Máquinas de Vectores Soporte. Se explica a continuación cómo afectó al análisis.

7.6.1 Análisis de Integración

Se comenzó a analizar el software con versiones de MailScanner (para Unix) anteriores a la 4.03. Este hecho es fundamental a la hora de integrar clasificadores externos a la herramienta, ya que en la versión 4.03 se produjo un cambio que provocó la modificación de la manera de incluir el nuevo clasificador de textos a MailScanner, como se explicará más adelante.

MailScanner es un software desarrollado en Perl, y su distribución la componen un conjunto de módulos distribuidos en varios ficheros. Entre estos módulos, MailScanner incorpora la herramienta SpamAssassin, y múltiples antivirus de libre distribución.

Se realizó un análisis exhaustivo de los ficheros que componen el software, para descubrir cuál era su funcionamiento, y en especial buscando el punto en el que MailScanner llamaba a SpamAssassin para realizar el chequeo del spam. En este punto debería haber una llamada a la función Checks del módulo SA, que recoge toda la herramienta SpamAssassin.

Una vez encontrado este punto del programa, sólo quedaría, por tanto, realizar a continuación la llamada al módulo que implementará el clasificador de textos externo.

La llamada al clasificador externo deberá realizarse en el mismo punto en el que se llama a SpamAssassin, ya que su propósito es exactamente el mismo: decidir si el correo es spam o no.

Se descubrió que en concreto, "Message.pm" era el módulo que realiza la comprobación de spam. Este módulo de perl contiene una subrutina llamada "IsSpam", que devuelve un valor binario que identifica si el correo es o no es spam.

El código de la rutina "IsSpam" comentada, se adjunta en el Anexo A. Se trata de una rutina que lleva a cabo, a grandes rasgos, el siguiente flujo:

- Inicialización de variables
- Escritura en ficheros de logs.
- Si el número de receptores del mensaje es menor que el máximo permitido para comprobar listas blancas: comprobar listas blancas.
    - Si se encuentra en listas blancas y no se quiere ejecutar SpamAssassin: salir y devolver 0 (no spam).
    - Si no, continuar.
- Comprobar listas negras.
    - Si se encuentra en listas negras: salir y devolver 1 (es spam).
    - Si no, continuar.
- Si no estaba en listas blancas, comprobar las listas negras en tiempo real.
- Si no se requiere el uso de SpamAssassin, devolver el resultado de la comprobación de las listas negras en tiempo real.
- Si se requiere el uso de SpamAssassin: ejecutar la subrutina "Checks" de SpamAssassin.
    - Guardar resultados de SpamAssassin.
    - El mensaje es spam si SpamAssassin lo dice, a menos que esté en listas blancas.
    - Añadir la cabecera de SpamAssassin al mensaje.
    - Escribir ficheros de log
    - Devolver resultado, 1 si es spam, 0 si no lo es.

Por tanto, la llamada al clasificador de textos externo habría que hacerla en esta subrutina justo antes, o después, de la llamada a SpamAssassin.

Al igual que ocurre con SpamAssassin, debería añadirse un parámetro en el fichero de configuración de MailScanner, para determinar

si se quiere o no que se ejecute este nuevo clasificador de textos. De esta forma se podrán evitar las dos ejecuciones, ejecutar uno de ellos, e incluso ejecutar los dos y comprobar los resultados de cada uno de ellos.

El nuevo módulo deberá situarse al mismo nivel que el módulo "SA.pm", de manera que la llamada será similar a la de SpamAssassin:

```
($SAsaysspam,$SAHighScoring,$saheader,$sascore,$salongreport)

   = MailScanner::SA::Checks($this);
```

7.6.2 MODIFICACIÓN DE MAILSCANNER.

Como se dijo en el apartado anterior, MailScanner añadió una mejora importante al sistema a partir de la versión 4.03. Desde esta versión en adelante, se proporciona un directorio en el que el usuario puede definir módulos, que se ejecutarán automáticamente por el mero hecho de estar dentro del directorio. Este directorio se llama CustomFunctions, y en la distribución se proporcionan dos ejemplos de posibles funciones.

La adición de esta nueva funcionalidad se puede observar en el fichero de cambios "ChangeLog":

```
New in Version 4.03

==================

- Added ability for you to be able to write your own
functions that produce values for configuration options, as well
as being able to use simple values or rulesets. Read
CustomConfig.pm and mailscanner.conf for more information on how
to do this.
```

A la hora de utilizar esta nueva funcionalidad, habrá que tener en cuenta ciertos parámetros de configuración incluidos en los ficheros "CustomConfig.pm" y "mailscanner.conf", que serán leídos por el nuevo módulo.

Cualquier módulo incluido en el directorio "CustomFunctions" podrá recibir un objeto Message y devolver los parámetros que se quieran.

Por tanto bastará con incluir en este directorio un módulo que incluya el clasificador de textos externo, que recibirá un mensaje y

devolverá un scoring que indique si el mensaje es spam o no, y que será añadido automáticamente por MailScanner a sus cabeceras.

MailScanner proporcionará como entrada al proceso los siguientes datos:

- Dirección IP de la conexión SMTP.
- Remitente del correo.
- Lista de destinatarios del correo, uno por línea.
- Una línea en blanco.
- El mensaje completo, incluyendo todas las cabeceras y el cuerpo en formato rfc822.

Las salidas que espera MailScanner son las siguientes:

- Un número entero o de coma flotante que proporcione el scoring de spam del mensaje.
- Un resumen de una línea que proporcione el resumen de los resultados del programa.

Los parámetros de configuración de MailScanner que habrá que configurar son los siguientes:

- **`Use Custom Spam Scanner = yes`** → Cuando se indica "yes" el detector de spam desarrollado se aplicará al mensaje. MailScanner incluye un temporizador de forma automática para que el programa no exceda el tiempo de ejecución permitido.
- **`Max Custom Spam Scanner Size = 20000`** → Indica el tamaño máximo del fragmento de mensaje que recibirá el programa.
- **`Custom Spam Scanner Timeout = 20`** → Indica el tiempo máximo en segundos permitido para que el detector de spam termine su ejecución. Si el programa no termina en este tiempo será matado automáticamente y generará un scoring 0.

- **`Max Custom Spam Scanner Timeouts = 10`** → Si el detector de spam falla más de este número de veces en una secuencia de mensajes, no se volverá a ejecutar hasta el próximo reinicio periódico de MailScanner. Este reinicio automático se configurará con el parámetro "Restart Every".

- **`Custom Spam Scanner Timeout History = 20`** → Representa la historia de timeouts del programa. Será el tamaño de la secuencia de mensajes bajo la cual aplica el parámetro anterior. Con los valores por defecto, si el programa incurre en timeout en 10 mensajes en una secuencia de 20 mensajes consecutivos, no se volverá a ejecutar hasta el siguiente reinicio de MailScanner.

7.6.3 EXPERIMENTO 6

En este experimento se ejecutó la prueba integrada de MailScanner con el clasificador SVM desarrollado en este proyecto.

Tal y como se explicó en el apartado anterior, la integración consistió en codificar el clasificador dentro de las CustomFunctions de MailScanner.

En el Anexo D se muestra el código fuente del "Custom Function" generado.

Además, para llevar a cabo una prueba integrada del flujo end to end, se desarrolló un pequeño programa en Perl para llevar a cabo el envío automático de varios correos a través de sendmail. Estos correos llegarán a MailScanner, que entre todos sus análisis ejecutará la Custom Function desarrollada. El código fuente para el envío automático de correos electrónicos se muestra en el Anexo E.

Se comprobó que los correos electrónicos llegan al nuevo módulo, se lleva a cabo el análisis y se devuelve a MailScanner el scoring que indica la probabilidad de que el correo sea Spam.

Este módulo utiliza los siguientes ficheros auxiliares:

- `.stopwords_all.txt`: listado de todas las stopwords, palabras que no se tendrán en cuenta en el análisis.

- `gsvc_classify`: ejecutable de GSVC que realizará la clasificación.

- `modelo.txt`: fichero que contendrá la descripción de la regla de clasificación de GSVC, utilizado por gsvc_classify.

- `.hashDF.txt`: fichero que contiene la información de ocurrencias de cada palabra en todos los documentos. Se actualizará con el análisis de cada correo electrónico.

- `.numeroMensajes.txt`: fichero que contiene un número, el número de mensajes totales para el cálculo de TFIDF. Se actualizará con el análisis de cada correo electrónico.

- `.features100000.txt`: listado de *features* ordenados por TFIDF que se tendrán en cuenta en la clasificación. Será recomienda su regeneración de forma periódica, por ejemplo cada semestre o cada año, junto con un reentrenamiento de la máquina.

- `.features100000_ord_TF.txt`: listado de *features* ordenados por TF que se tendrán en cuenta en la clasificación. Será recomienda su regeneración de forma periódica, por ejemplo cada semestre o cada año, junto con un reentrenamiento de la máquina.

Además, es necesario incluir los siguientes parámetros en el fichero de configuración de MailScanner:

- `Numfeatures:` número de *features* que se tendrán en cuenta en el análisis.

- `tipoAnalisis:` tipo de análisis que se ejecutará: TF ó TFIDF.

8 CONCLUSIONES

A partir del estudio profundo de métodos y algoritmos de clasificación de textos, y de los desarrollos y experimentos realizados, puedo extraer las siguientes conclusiones:

- El aprendizaje máquina es un muy buen método de clasificación de textos. Tanto el análisis Bayesiano como el introducido y desarrollado en este proyecto, basado en Máquinas de Vectores Soporte, consiguen resultados muy buenos en torno al 99,5% de acierto.

- Como detector de spam, tanto el análisis Bayesiano como las Máquinas de Vectores Soporte han demostrado ser los complementos perfectos para las reglas y métodos deterministas implementados por MailScanner.

- Es fundamental realizar un buen entrenamiento para que la máquina de vectores soporte consiga resultados óptimos:

    - El conjunto de datos tiene que ser grande, del orden de miles de documentos.

    - Debe componerse una buena lista de stopwords, o palabras que no se tendrán en cuenta en los análisis por no ser descriptivas (pronombres, preposiciones, conjunciones…, etiquetas html…).

    - El proceso de selección de *features* es crítico en el proceso global, y es fundamental elegir correctamente el número de *features* que deben tenerse en cuenta en cada caso concreto. Probablemente los correos electrónicos recibidos en el servidor de la universidad tendrán características distintas a los recibidos en una empresa o en una cuenta particular. Será necesario optimizar el listado de *features* para cada entorno.

    - El orden de los correos electrónicos en la entrada del proceso de entrenamiento es muy importante. El

entrenamiento debe recibir los correos etiquetados de forma aleatoria para poder obtener una buena regla de clasificación. Si el proceso recibe primero todos los correos spam y luego todos los correos ham (o viceversa), no se obtendrá una buena regla de clasificación, llegando a tener resultados casi aleatorios. Esto apoya la condición de independencia de los datos asumida en el análisis teórico del clasificador.

- El clasificador GSVC, que implementa un algoritmo incremental iterativo, y que ha sido definido y desarrollado por compañeros de la universidad ha demostrado ser un excelente clasificador, con muy buenas prestaciones en cuanto a aprendizaje y en cuanto a consumo de recursos.
- La herramienta MailScanner es un producto muy potente de análisis de correo electrónico, y da la posibilidad de integrar módulos externos de forma muy sencilla.

9 AGRADECIMIENTOS

Quiero agradecer a todas las personas que me han apoyado y que de alguna forma han hecho posible que este proyecto saliera adelante. Gracias a mi familia, mis amigos, a mis compañeros de clase con los que tantos momentos buenos y no tan buenos compartí durante la carrera. Pero en concreto haré mención especial de aquellos que han sido fundamentales para la finalización de este proyecto.

En primer lugar, mi amigo y compañero de tantas y tantas fatigas durante la carrera, con el que he compartido muchos momentos dentro y fuera de la universidad. Gracias Juan, por todo tu apoyo, por ser pesado conmigo y estar siempre pendiente para que no abandonara. Sin tu ayuda todo habría sido mucho más difícil.

Gracias a mi chica, Mayte, por haber sacrificado sus vacaciones ayudándome y dejándome todo el tiempo para acabar el proyecto. Por darme el cariño y los ánimos necesarios día a día.

También gracias a mis padres y a mi hermana, que estaban tanto o más preocupados que yo por tener esta tarea pendiente. Gracias por dármelo todo y hacerme la vida mucho más fácil.

Y gracias a Harold, el director de mi proyecto, por confiar en mí y por su enorme paciencia conmigo.

Gracias a todos!

10 BIBLIOGRAFÍA

- Alan Schwartz – "SpamAssassin" 2004 [eBook], O'Reilly
- Julian Field – "MailScanner, Open Source Secure Mail Gateway Software. Administrators Guide, version 1.0.1"
- Larry Wall, Tom Christiansen, Randal L. Schwartz – "Programming Perl". O'Reilly.
- Randal L. Schwartz – "Learning Perl"
- "MailScanner.conf Reference Manual" V1.1 (MailScanner V4.42.9)
- MailScanner, Open source Secure Mail Gateway Software. Administrators Guide", Version 1.0.1
- "Sendmail Installation and Operation Guide"
- Nello Cristianini, John Shawe-Taylor – "An introduction to Support Vector Machines and other kernel-based learning methods", 2000. Cambridge University Press.
- E. Parrado Hernández, I. Mora Jiménez, J. Arenas García, A.R. Figueiras Vidal, A. Navia Vázquez – "Growing support vector classifiers with controlled complexity". 2002. Pattern Recognition. The journal of the pattern recognition society.
- Theodoros Evgeniou, Massimiliano Pontil – "Statistical Learning Theory: a Primer". Kluwer Academic Publishers, Boston.
- O.L. Mangasarian – "Data Mining via Support Vector Machines". University of Wisconsin.
- Chih-Wei Hsu, Chih-Chung Chang, Chig-Jen Lin – "A practical guide to Support Vector Classification". National Taiwan University.
- Chih-Chung Chang, Chih-Jen Lin – "LIBSVM: a library for Support Vector Machines". National Taiwan University.

- Rong-En Fan, Pai-Hsuen Chen, Chih-Jen Lin – “Working set selection using second order information for training Support Vector Machines”. National Taiwan University.
- Thorsten Joachims – “Making Large-Scale SVM Learning Practical”. Universität Dortmund.
- Thorsten Joachims – “Transductive Inference for Text Classification using Support Vector Machines”. Universität Dortmund.
- Thorsten Joachims – “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”. Universität Dortmund.
- Nello Cristianini – “Kernel Methods for Text Analysis”
- Nello Cristianini – “Kernel Methods for Pattern Analysis”
- Bernhard Schölkopf – “SVM and Kernel Methods”. Max-Planck-Institud für biologische Kybernetik.
- Thorsten Joachims – “Learning to classify text using support vector machines: methods, theory, and algorithms”. Kluwer Academic Publishers.

11 ANEXO A – SUBRUTINA ISSPAM DEL MÓDULO MESSAGE.PM

```
# Is this message spam? Try to build the spam report and store it in
# the message.
sub IsSpam {
  my $this = shift;
  my($includesaheader, $iswhitelisted);
  my $spamheader    = "";
  my $rblspamheader = "";
  my $saspamheader  = "";
  my $RBLsaysspam   = 0;
  my $rblcounter    = 0;
  my $LogSpam = MailScanner::Config::Value('logspam');
  my $LogNonSpam = MailScanner::Config::Value('lognonspam');
  my $LocalSpamText = MailScanner::Config::LanguageValue($this, 'spam');

  # Construct a pretty list of all the unique domain names for logging
  my(%todomain, $todomain);
  foreach $todomain (@{$this->{todomain}}) {
    $todomain{$todomain} = 1;
  }
  $todomain = join(',', keys %todomain);
  my $recipientcount = @{$this->{to}};

  # $spamwhitelisted      set by IsSpam
  # $spamblacklisted      set by IsSpam
  # $isspam               set by IsSpam
  # $ishigh               set by IsSpam
  # $spamreport           set by IsSpam

  $this->{spamwhitelisted} = 0;
```

```
        $this->{spamblacklisted} = 0;
        $this->{isspam} = 0;
        $this->{ishigh} = 0;
        $this->{spamreport} = "";
        $this->{sascore} = 0;

        ## If it's a blacklisted address, don't bother doing any checks at all
        #if (MailScanner::Config::Value('spamblacklist', $this)) {
        #  $this->{isspam} = 1;
        #  $this->{spamreport} = 'spam (blacklisted)';
        #  MailScanner::Log::InfoLog("Message %s from %s (%s) " .
        #                            " is spam (blacklisted)",
        #                            $this->{id}, $this->{clientip},
        #                            $this->{from});
        #  return 1;
        #}

        # Work out if they always want the SA header
        $includesaheader = MailScanner::Config::Value('includespamheader',
$this);

        # Do the whitelist check before the blacklist check.
        # If anyone whitelists it, then everyone gets the message.
        # If no-one has whitelisted it, then consider the blacklist.
        $iswhitelisted = 0;
        my $maxrecips = MailScanner::Config::Value('whitelistmaxrecips');
        $maxrecips = 999999 unless $maxrecips;

        if ($recipientcount<=$maxrecips) {
          if (MailScanner::Config::Value('spamwhitelist', $this)) {
            # Whitelisted, so get out unless they want SA header
            #print STDERR "Message is whitelisted\n";
```

```
            MailScanner::Log::InfoLog("Message %s from %s (%s) is
whitelisted",
                                      $this->{id}, $this->{clientip}, $this-
>{from})
              if $LogSpam || $LogNonSpam;
            $iswhitelisted = 1;
            $this->{spamwhitelisted} = 1;
            # whitelisted and doesn't want SA header so get out
            return 0 unless $includesaheader;
          }
        } else {
          # Had too many recipients, ignoring the whitelist
          MailScanner::Log::InfoLog("Message %s from %s (%s) ignored
whitelist, " .
                                    "had %d recipients (>%d)", $this->{id},
                                    $this->{clientip}, $this->{from},
                                    $recipientcount, $maxrecips)
            if $LogSpam || $LogNonSpam;
        }

        # If it's a blacklisted address, don't bother doing any checks at all
        if (MailScanner::Config::Value('spamblacklist', $this)) {
          $this->{spamblacklisted} = 1;
          $this->{isspam} = 1;
          $this->{ishigh} = 1
            if MailScanner::Config::Value('blacklistedishigh', $this);
          $this->{spamreport} = $LocalSpamText . ' (' .
                          MailScanner::Config::LanguageValue($this,
'blacklisted') .
                          ')';
          MailScanner::Log::InfoLog("Message %s from %s (%s) to %s" .
                                    " is spam (blacklisted)",
                                    $this->{id}, $this->{clientip},
```

```
                                $this->{from}, $todomain)
            if $LogSpam;
          return 1;
        }


        if (!$iswhitelisted) {
          # Not whitelisted, so do the RBL checks
          #$rblspamheader     = MailScanner::RBLs::Checks($this);
          ($rblcounter, $rblspamheader) = MailScanner::RBLs::Checks($this);
          $RBLsaysspam       = 1 if $rblcounter;
          #$RBLsaysspam       = 1 if $rblspamheader;
          # Add leading "spam, " if RBL says it is spam. This will be at the
          # front of the spam report.
          $rblspamheader     = $LocalSpamText . ', ' . $rblspamheader if
$rblcounter;
          $this->{isspam}    = 1 if $rblcounter;
          $this->{isrblspam} = 1 if $rblcounter;
          $this->{ishigh}    = 1 if $rblcounter >= MailScanner::Config::Value(
                                                  'highrbls', $this);
          #print STDERR "RBL report is \"$rblspamheader\"\n";
          #print STDERR "RBLCounter = $rblcounter\n";
          #print STDERR "HighRBLs   = " .
          #             MailScanner::Config::Value('highrbls', $this) . "\n";
        }


        # Don't do the SA checks if they have said no.
        unless (MailScanner::Config::Value('usespamassassin', $this)) {
          $this->{spamwhitelisted} = $iswhitelisted;
          $this->{spamreport}      = $rblspamheader;
          MailScanner::Log::InfoLog("Message %s from %s (%s) to %s is %s",
                                    $this->{id}, $this->{clientip},
                                    $this->{from}, $todomain, $rblspamheader)
```

```
    if $RBLsaysspam && $LogSpam;
  return $RBLsaysspam;
}

# If it's spam and they dont want to check SA as well
if ($this->{isspam} &&
    !MailScanner::Config::Value('checksaifonspamlist', $this)) {
  $this->{spamwhitelisted} = $iswhitelisted;
  $this->{spamreport}      = $rblspamheader;
  MailScanner::Log::InfoLog("Message %s from %s (%s) to %s is %s",
                            $this->{id}, $this->{clientip},
                            $this->{from}, $todomain, $rblspamheader)
    if $RBLsaysspam && $LogSpam;
  return $RBLsaysspam;
}

# They must want the SA checks doing.

my $SAsaysspam = 0;
my $SAHighScoring = 0;
my $saheader = "";
my $sascore  = 0;
my $salongreport = "";
($SAsaysspam, $SAHighScoring, $saheader, $sascore, $salongreport)
  = MailScanner::SA::Checks($this);
$this->{sascore} = $sascore; # Save the actual figure for use later...
# Trim all the leading rubbish off the long SA report and turn it back
# into a multi-line string, then store it in the message properties.
$salongreport =~ s/^.* pts rule name/ pts rule name/;
$salongreport =~ tr/\0/\n/;
$this->{salongreport} = $salongreport;
```

```
        #print STDERR $salongreport . "\n";


        # Fix the return values
        $SAsaysspam = 0 unless $saheader;    # Solve bug with empty SAreports
        $saheader =~ s/\s+$//g if $saheader; # Solve bug with trailing space


        #print STDERR "SA report is \"$saheader\"\n";
        #print STDERR "SAsaysspam = $SAsaysspam\n";
        $saheader = MailScanner::Config::LanguageValue($this, 'spamassassin')
.
                    " ($saheader)" if $saheader;


        # The message really is spam if SA says so (unless it's been
whitelisted)
        unless ($iswhitelisted) {
          $this->{isspam} |= $SAsaysspam;
          $this->{issaspam} = $SAsaysspam;
        }


        # If it's spam...
        if ($this->{isspam}) {
          #print STDERR "It is spam\nInclude SA = $includesaheader\n";
          #print STDERR "SAHeader = $saheader\n";
          $spamheader = $rblspamheader;
          # If it's SA spam as well, or they always want the SA header
          if ($SAsaysspam || $includesaheader) {
            #print STDERR "Spam or Add SA Header\n";
            $spamheader = $LocalSpamText unless $spamheader;
            $spamheader .= ', ' if $spamheader && $saheader;
            $spamheader .= $saheader;
            $this->{ishigh} = 1 if $SAHighScoring;
          }
```

```
        } else {
          # It's not spam...
          #print STDERR "It's not spam\n";
          #print STDERR "SAHeader = $saheader\n";
          $spamheader = MailScanner::Config::LanguageValue($this, 'notspam');
          if ($iswhitelisted) {
            $spamheader .= ' (' .
                          MailScanner::Config::LanguageValue($this,
'whitelisted') .
                          ')';
          }
          # so RBL report must be blank as you can't force inclusion of that.
          # So just include SA report.
          $spamheader .= ", $saheader";
        }

        # Now just reflow and log the results
        if ($spamheader ne "") {
          $spamheader = $this->ReflowHeader(
                        MailScanner::Config::Value('spamheader',$this),
$spamheader);
          $this->{spamreport} = $spamheader;
        }

        # Do the spam logging here so we can log high-scoring spam too
        if (($LogSpam && $this->{isspam}) || ($LogNonSpam && !$this-
>{isspam})) {
          my $ReportText = $spamheader;
          $ReportText =~ s/\s+/ /sg;
          MailScanner::Log::InfoLog("Message %s from %s (%s) to %s is %s",
                                    $this->{id}, $this->{clientip},
                                    $this->{from}, $todomain, $ReportText);
        }
```

```
  return $this->{isspam};
}
```

12 ANEXO B – CÓDIGO FUENTE EXTRACCIÓN DE *FEATURES*

```
use strict;

use Mail::MboxParser;

use HTML::Tagset;

imprimeHora("Inicio Script\n");

open(STOPWORDS,"../../stopwords_all.txt") || die "No se pudo abrir el fichero\n";

my %stopwords;

while (<STOPWORDS>)

{

chop($_);

$stopwords{$_}=1;

}

#print "\nLas stopwords son:\n";

#print keys %stopwords;

#print "\n";

imprimeHora("Se han leido las stopwords\n");

imprimeHora("Se annaden las etiquetas html\n");

%stopwords=(%stopwords,%HTML::Tagset::isKnown);

my %hash_TF_total;

my %hash_DF;

my $num_mensajes = 0;

my $mbSPAM = Mail::MboxParser->new('buzones_spam/TRAIN', decode=>'ALL');

my $mbHAM = Mail::MboxParser->new('buzones_ham/TRAIN', decode=>'ALL');

imprimeHora("Se procede a leer el mbox\n");


my $iteracion=0;

my $mailbox;

imprimeHora("Comienza parseo del spam");

parseaMailBox($mbSPAM);

imprimeHora("Terminado mbSPAM. Se vacía.iteracion:".$iteracion);
```

```
$mbSPAM="";
parseaMailBox($mbHAM);
imprimeHora("Terminado mbHAM. Se vacía.iteracion:".$iteracion);
$mbHAM="";

sub parseaMailBox{
my $mailbox=shift;
imprimeHora("Comienza parseaMailBox. Iteracion numero:".$iteracion);
BUCLE: while( my $msg = $mailbox->next_message )
{
$iteracion+=1;

if ($iteracion%1000==0)
{
       imprimeHora("Mensaje numero ".$iteracion."\n");
}
my $cuerpo = $msg->body($msg->find_body);
my $texto=$cuerpo->as_string ([strip_sig => 1]);
my $longitudtexto=length($texto);
if ($longitudtexto>=1000000)
{
        print $iteracion."longitudtexto:".length($texto)."\n";
        print "Se salta este correo\n";
$iteracion-=1;
        next BUCLE;
}
## Quito acentos y dieresis ##
$texto = &sustituyeVocales($texto);
## Paso a minusculas ##
$texto=lc $texto;
my @palabras_correo;
```

```
@palabras_correo = split(/\s+|\W+/,$texto);
my $palabra;
my %hash_local=[];
foreach $palabra (@palabras_correo)
{
	if ( (exists $stopwords{$palabra}) || ($palabra eq ""))
	{
		next;
	}
	else
	{
		if ( exists $hash_local{$palabra} )
		{
			$hash_local{$palabra} += 1;
		}
		else
		{
			$hash_local{$palabra} = 1;
		}
	}
}
my $termino;
foreach $termino (keys %hash_local)
{
	if (exists $hash_TF_total{$termino})
	{
		$hash_TF_total{$termino}+=$hash_local{$termino};
		$hash_DF{$termino}+=1;
	}
	else
	{
```

```
                $hash_TF_total{$termino}=$hash_local{$termino};
                $hash_DF{$termino}=1;
            }
        }
        $msg="";
        $cuerpo="";
        $texto="";
        $palabra="";
        $termino="";
        @palabras_correo=();
        %hash_local=[];
        }
        $mailbox="";
        imprimeHora("Finaliza parseaMailBox. Iteracion numero:".$iteracion);
        my %hash_TF_IDF;
        my $term_temp;
        my $suma=0;
        $num_mensajes=$iteracion;
        imprimeHora("Calculando productos y suma\n");
        foreach $term_temp (keys %hash_TF_total)
        {
        my
$producto=$hash_TF_total{$term_temp}*log($num_mensajes/$hash_DF{$term_temp})/l
og(10);
        $hash_TF_IDF{$term_temp}=$producto;
        $suma+=($producto)**2;
        }
        my $raiz=sqrt($suma);
        my $term_temp2;
        foreach $term_temp2 (keys %hash_TF_IDF)
        {
        $hash_TF_IDF{$term_temp2}=$hash_TF_IDF{$term_temp2}/$raiz;
```

```
}


sub porTFIDFdescendente {
        $hash_TF_IDF{$b} <=> $hash_TF_IDF{$a};
}

sub porTFdescendente {
$hash_TF_total{$b} <=> $hash_TF_total{$a};
}

my @ordenados = sort porTFIDFdescendente (keys %hash_TF_IDF);
my @ordenados_TF = sort porTFdescendente (keys %hash_TF_total);
open SALIDA100, ">features100.txt";
open SALIDA1000, ">features1000.txt";
open SALIDA10000, ">features10000.txt";
open SALIDA100000, ">features100000.txt";
open SALIDA100_ord_TF, ">features100_ord_TF.txt";
open SALIDA1000_ord_TF, ">features1000_ord_TF.txt";
open SALIDA10000_ord_TF, ">features10000_ord_TF.txt";
open SALIDA100000_ord_TF, ">features100000_ord_TF.txt";
open SALIDA100_TFIDF, ">features100_TFIDF.txt";
open SALIDA1000_TFIDF, ">features1000_TFIDF.txt";
open SALIDA10000_TFIDF, ">features10000_TFIDF.txt";
open SALIDA100000_TFIDF, ">features100000_TFIDF.txt";
open SALIDA100_TF, ">features100_TF.txt";
open SALIDA1000_TF, ">features1000_TF.txt";
open SALIDA10000_TF, ">features10000_TF.txt";
open SALIDA100000_TF, ">features100000_TF.txt";
my $temp;
imprimeHora("Escribiendo features");
```

```
        print SALIDA100 (join(" ",@ordenados[0 .. 99])."\n");
        print SALIDA1000 (join(" ",@ordenados[0 .. 999])."\n");
        print SALIDA10000 (join(" ",@ordenados[0 .. 9999])."\n");
        print SALIDA100000 (join(" ",@ordenados[0 .. 99999])."\n");
        print SALIDA100_ord_TF (join(" ",@ordenados_TF[0 .. 99])."\n");
        print SALIDA1000_ord_TF (join(" ",@ordenados_TF[0 .. 999])."\n");
        print SALIDA10000_ord_TF (join(" ",@ordenados_TF[0 .. 9999])."\n");
        print SALIDA100000_ord_TF (join(" ",@ordenados_TF[0 .. 99999])."\n");
        imprimeHora("Escribiendo features con valores de TFIDF");
        my $num_lin;
        $num_lin=0;
        foreach $temp (@ordenados[0 .. 99999])
        {
            $num_lin+=1;
            if ($num_lin<=100){
                print SALIDA100_TFIDF $temp." => ".$hash_TF_IDF{$temp}."\n";
                print SALIDA1000_TFIDF $temp." => ".$hash_TF_IDF{$temp}."\n";
                print SALIDA10000_TFIDF $temp." => ".$hash_TF_IDF{$temp}."\n";
                print SALIDA100000_TFIDF $temp." => ".$hash_TF_IDF{$temp}."\n";
            }else{
                if($num_lin<=1000){
                    print          SALIDA1000_TFIDF          $temp."          =>
".$hash_TF_IDF{$temp}."\n";
                    print          SALIDA10000_TFIDF          $temp."          =>
".$hash_TF_IDF{$temp}."\n";
                    print          SALIDA100000_TFIDF          $temp."          =>
".$hash_TF_IDF{$temp}."\n";
                }else{
                    if($num_lin<=10000){
                        print         SALIDA10000_TFIDF         $temp."         =>
".$hash_TF_IDF{$temp}."\n";
                        print         SALIDA100000_TFIDF         $temp."         =>
".$hash_TF_IDF{$temp}."\n";
                    }else{
```

```
                    if($num_lin<=100000){

                        print      SALIDA100000_TFIDF      $temp."      =>
".$hash_TF_IDF{$temp}."\n";

                    }

                }

            }

        }

        imprimeHora("Escribiendo features con valores de TF");

        $num_lin=0;

        foreach $temp (@ordenados_TF[0 .. 99999])

        {

            $num_lin+=1;

            if ($num_lin<=100){

                print SALIDA100_TF $temp." => ".$hash_TF_total{$temp}."\n";

                print SALIDA1000_TF $temp." => ".$hash_TF_total{$temp}."\n";

                print SALIDA10000_TF $temp." => ".$hash_TF_total{$temp}."\n";

                print SALIDA100000_TF $temp." => ".$hash_TF_total{$temp}."\n";

            }else{

                if($num_lin<=1000){

                    print SALIDA1000_TF $temp." => ".$hash_TF_total{$temp}."\n";

                    print          SALIDA10000_TF          $temp."          =>
".$hash_TF_total{$temp}."\n";

                    print          SALIDA100000_TF          $temp."          =>
".$hash_TF_total{$temp}."\n";

                }else{

                    if($num_lin<=10000){

                        print         SALIDA10000_TF         $temp."         =>
".$hash_TF_total{$temp}."\n";

                        print         SALIDA100000_TF         $temp."         =>
".$hash_TF_total{$temp}."\n";

                    }else{

                        if($num_lin<=100000){
```

```
                        print SALIDA100000_TF $temp." => ".$hash_TF_total{$temp}."\n";
                    }
                }
            }
        }
    }

    imprimeHora("Fin del script");

    sub sustituyeVocales {
    my $var = $_[0];
    $var =~ s/á/a/g;
    $var =~ s/é/e/g;
    $var =~ s/í/i/g;
    $var =~ s/ó/o/g;
    $var =~ s/ú/u/g;
    $var =~ s/ü/u/g;
    $var =~ s/Á/A/g;
    $var =~ s/É/E/g;
    $var =~ s/Í/i/g;
    $var =~ s/Ó/O/g;
    $var =~ s/Ú/U/g;
    $var =~ s/Ü/U/g;
    return $var;
    }

    sub imprimeHora {
    my $texto = $_[0];
    my $mensaje_log;
    my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst);
    ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
```

```
	$mensaje_log		=		($mday."-".$mon."-".($year+1900)."
".$hour.":".$min.":".$sec." ".$texto."\n");

	print $mensaje_log;

	}
```

13 ANEXO C – CÓDIGO FUENTE GENERACIÓN FICHEROS TRAIN Y TEST

```
use strict;

use Mail::MboxParser;

use HTML::Tagset;


print `date`;


my $numfeatures = shift;

my $tipoAnalisis = shift;

if ($tipoAnalisis ne "TF" && $tipoAnalisis ne "TFIDF")

{

       print "Formato incorrecto\nEjecucion correcta:\n\n";

       print "perl genera_test_train.pl NUMERODEFEATURES TIPOANALISIS\n\n";

       print

         "donde NUMEROFEATURES es un entero positivo \n";

    print "y TIPOANALISIS puede tomar los valores: TF, TFIDF\n";

       exit 1;

}

if ( $tipoAnalisis eq "TF")

{

       open( FEATURES, "features100000_ord_TF.txt" ) || die "No se pudo abrir
el fichero de features100000_ord_TF\n";

        imprimeHora("se abre el fichero de features100000_ord_TF.txt");

}

else

{

       open( FEATURES, "features100000.txt" ) || die "No se pudo abrir el
fichero de features100000\n";

        imprimeHora("se abre el fichero de features100000.txt");

}


my @array_features_t;
```

```
my @array_features;

while (<FEATURES>) {

	my $var = $_;

	@array_features_t = split(/ /,$var);

}

@array_features = @array_features_t[0..($numfeatures-1)];

@array_features_t = [];

my $tempo;

my %hash_features_orden;

my %hash_features;

my $contador = 1;

foreach $tempo (@array_features) {

	$hash_features_orden{$tempo} = $contador;

	$hash_features{$tempo}	= 0;

	$contador += 1;

}


open(STOPWORDS,"../../stopwords_all.txt") || die "No se pudo abrir el fichero
de stopwords\n";

my %stopwords;

while (<STOPWORDS>) {

	chop($_);

	$stopwords{$_} = 1;

}

imprimeHora("Se han leido las stopwords");

my %hash_DF = %hash_features;

my $num_mensajes = 0;

my $num_ham = 0;

my $num_spam = 0;

my $mbSpamTrain = Mail::MboxParser->new( '../caso5/buzones_spam/TRAIN', decode
=> 'ALL' );

my $mbSpamTest = Mail::MboxParser->new( '../caso5/buzones_spam/TEST', decode
=> 'ALL' );

my $mbHamTrain = Mail::MboxParser->new( '../caso5/buzones_ham/TRAIN', decode
=> 'ALL' );
```

```
my $mbHamTest = Mail::MboxParser->new( '../caso5/buzones_ham/TEST', decode =>
'ALL' );


my $num_ham_train=0;

my $num_ham_test=0;

my $num_spam_train=0;

my $num_spam_test=0;

#imprimeHora("Se han abierto los mboxes, que contienen ".$num_mensajes."
mensajes");

#imprimeHora("num_ham_train".$num_ham_train);

#imprimeHora("num_ham_test".$num_ham_test);

#imprimeHora("num_spam_train".$num_spam_train);

#imprimeHora("num_spam_test".$num_spam_test);

imprimeHora("Se procede a leer el mbox");

my @mboxes_ham = ($mbHamTrain, $mbHamTest);

my @mboxes_spam = ($mbSpamTrain, $mbSpamTest);

my $iteracion = 0;

my $mailbox_ham;

my $mailbox_spam;


open( TFHAM,  ">".$tipoAnalisis."ham".$numfeatures.".txt" );

open( TFSPAM, ">".$tipoAnalisis."spam".$numfeatures.".txt" );

my $TrainTest=0;

foreach $mailbox_ham (@mboxes_ham) {

       $TrainTest=$TrainTest+1;

       BUCLEHAM: while ( my $msg = $mailbox_ham->next_message ) {

              $iteracion += 1;

              if ( $iteracion % 1000 == 0 ) {

                     imprimeHora("Mensaje numero " . $iteracion." de HAM");

              }

              my $cuerpo = $msg->body( $msg->find_body );

              my $texto = $cuerpo->as_string( [ strip_sig => 1 ] );

              my $longitudtexto=length($texto);

        if ($longitudtexto>=1000000)

        {
```

```
    print $iteracion."longitudtexto:".length($texto)."\n";
    print "Se salta este correo\n";
    next BUCLEHAM;
}
     ## Quito acentos y dieresis ##
     $texto = &sustituyeVocales($texto);
     ## Paso a minusculas ##
     $texto = lc $texto;
     my @palabras_correo;
     @palabras_correo = split( /\s+|\W+/, $texto );
     my $palabra;
     my %hash_local;

     if ( $iteracion % 1000 == 0 ) {
            imprimeHora("Extrayendo palabras de correo...");
     }
     foreach $palabra (@palabras_correo) {
    if ( ( exists $stopwords{$palabra} ) || ( $palabra eq "" ) ) {
                  next;
            }
            else {
        if ( ( exists $hash_features{$palabra} ) ) {
            if ( exists $hash_local{$palabra} ) {
                $hash_local{$palabra} += 1;
            }
            else {
                $hash_local{$palabra} = 1;
            }
        }
        else {
            next;
        }
    }
     }
```

```
my $termino;
if ( $iteracion % 1000 == 0 ) {
	imprimeHora("Actualizando hashes TF y DF...");
}

foreach $termino (@array_features){
	if ( exists $hash_local{$termino} )
	{
		print TFHAM $hash_local{$termino}." ";
	}
	else
	{
		print TFHAM "0 ";
	}
}

foreach $termino ( keys %hash_local ) {
	$hash_DF{$termino} += 1;
#	print TFHAM $hash_local{$termino} . " ";
}

#		my $tf_temp;
#		for $tf_temp (@array_features){
#			print TFHAM $hash_local{$tf_temp}." ";
#		}
print TFHAM "\n";

if($TrainTest==1)
{
	$num_ham_train=$num_ham_train+1;
}
else
{
	if($TrainTest==2)
```

```
                        {
                                $num_ham_test=$num_ham_test+1;
                        }
                }
        }
}
$num_ham=$num_ham_train+$num_ham_test;
imprimeHora("Numero de mensajes HAM:".$num_ham);
close(TFHAM);
$TrainTest=0;
foreach $mailbox_spam (@mboxes_spam) {
        $TrainTest=$TrainTest+1;
        BUCLESPAM: while ( my $msg = $mailbox_spam->next_message ) {
                $iteracion += 1;
                if ( $iteracion % 1000 == 0 ) {
                        imprimeHora("Mensaje numero " . $iteracion." de SPAM");
                }
                my $cuerpo = $msg->body( $msg->find_body );
                my $texto = $cuerpo->as_string( [ strip_sig => 1 ] );
                my $longitudtexto=length($texto);
          if ($longitudtexto>=1000000)
          {
              print $iteracion."longitudtexto:".length($texto)."\n";
              print "Se salta este correo\n";
              next BUCLESPAM;
          }
                ## Quito acentos y dieresis ##
                $texto = &sustituyeVocales($texto);
                ## Paso a minusculas ##
                $texto = lc $texto;
                my @palabras_correo;
                @palabras_correo = split( /\s+|\W+/, $texto );
                my $palabra;
                my %hash_local;
```

```
		if ( $iteracion % 1000 == 0 ) {
			imprimeHora("Extrayendo palabras de correo...");
		}
		foreach $palabra (@palabras_correo) {
			if ( ( exists $stopwords{$palabra} ) || ( $palabra eq "" )
) {
				next;
			}
			else {
				if ( ( exists $hash_features{$palabra} ) ) {
					if ( exists $hash_local{$palabra} ) {
						$hash_local{$palabra} += 1;
					}
					else {
						$hash_local{$palabra} = 1;
					}
				}
				else {
					next;
				}
			}
		}
		my $termino;
		if ( $iteracion % 1000 == 0 ) {
			imprimeHora("Actualizando hashes TF y DF...");
		}

		foreach $termino (@array_features){
			if ( exists $hash_local{$termino} )
			{
				print TFSPAM $hash_local{$termino}." ";
			}
			else
```

```
                {
                        print TFSPAM "0 ";
                }
        }

        foreach $termino ( keys %hash_local ) {
                $hash_DF{$termino} += 1;
        #       print TFSPAM $hash_local{$termino} . " ";
        }

        #               my $tf_temp;
        #               for $tf_temp (@array_features){
        #                       print TFSPAM $hash_local{$tf_temp}." ";
        #               }
        print TFSPAM "\n";

        if($TrainTest==1)
        {
                $num_spam_train=$num_spam_train+1;
        }
        else
        {
                if($TrainTest==2)
                {
                        $num_spam_test=$num_spam_test+1;
                }
        }
    }

}
$num_spam=$num_spam_train+$num_spam_test;
imprimeHora("Numero de mensajes SPAM:".$num_spam);
$num_mensajes=$num_ham+$num_spam;
close(TFSPAM);
```

```
open(TFHAMLECTURA, $tipoAnalisis."ham".$numfeatures.".txt" );
open(TFSPAMLECTURA, $tipoAnalisis."spam".$numfeatures.".txt" );
if ($tipoAnalisis eq "TF"){
    open(TFTRAIN,">".$tipoAnalisis."train".$numfeatures.".txt" );
    open(TFTEST,">".$tipoAnalisis."test".$numfeatures.".txt" );
}else{
    open(TFIDFTRAIN,">".$tipoAnalisis."train".$numfeatures.".txt" );
    open(TFIDFTEST,">".$tipoAnalisis."test".$numfeatures.".txt" );
}

#Se guarda en fichero el numero de mensajes
open(NUMMENSAJES,">numeroMensajes.txt");
print NUMMENSAJES $num_mensajes;
close(NUMMENSAJES);

#Se guarda en fichero el hash de Document Frequency
my $palabraDF;
open(HASHDF,">hashDF.txt");
foreach $palabraDF (@array_features){
      print HASHDF $palabraDF." ".$hash_DF{$palabraDF}."\n";
}
close(HASHDF);

my $train_test=0;
imprimeHora("se va a recorrer el fichero tfham");
while (<TFHAMLECTURA>) {

      my @tf_ham = split(/ /, $_ );
      my @solo_tf_ham = @tf_ham;
      my %hash_TF_IDF;
      my $term_temp;
      my $suma = 0;
      #imprimeHora("Calculando productos y suma");
```

```
        my $cont = 0;
        foreach $term_temp (@array_features) {

                #       imprimeHora("Log 1: ".$term_temp);
                #       imprimeHora("Log 2: ".$tf_ham[$cont]);
                my $producto;
                if ( $hash_DF{$term_temp} != 0 ) {
                        $producto=$tf_ham[$cont]*log(
$num_mensajes/$hash_DF{$term_temp})/log(10);
                }
                else {
                        $producto = 0;
                }

                #       print "Log 3: ".$producto."\n";
                $tf_ham[$cont] = $producto;
                $suma += ($producto)**2;

                #       print "Log 4: ".$suma."\n";
                $cont += 1;
        }
        my $raiz = sqrt($suma);

        #print "Log 5: ".$raiz."\n";
        my $term_temp2;
        my $cont2 = 0;
        if ($train_test < $num_ham_train){
               if ($tipoAnalisis eq "TF"){
              print TFTRAIN "-1 ";
               }
               else{
              print TFIDFTRAIN "-1 ";
               }
```

```
	}
	else{
		if ($tipoAnalisis eq "TF"){
		print TFTEST "-1 ";
		}
		else{
		print TFIDFTEST "-1 ";
		}
	}
	foreach $term_temp2 (@array_features) {
		if ( $raiz == 0 ) {
			$tf_ham[$cont2] = 0;
		}
		else {
			$tf_ham[$cont2] = $tf_ham[$cont2] / $raiz;
		}
		if ( $tf_ham[$cont2] != 0 ) {
			if ($train_test < $num_ham_train){
				if ($tipoAnalisis eq "TF"){
			print TFTRAIN $hash_features_orden{$term_temp2} . ":" .
$solo_tf_ham[$cont2] . " ";
				}
				else{
			print TFIDFTRAIN $hash_features_orden{$term_temp2} . ":" .
$tf_ham[$cont2] . " ";
				}
			}
			else{
				if ($tipoAnalisis eq "TF"){
			print TFTEST $hash_features_orden{$term_temp2} . ":" .
$solo_tf_ham[$cont2] . " ";
				}
				else{
			print TFIDFTEST $hash_features_orden{$term_temp2} . ":" .
$tf_ham[$cont2] . " ";
				}
```

```
                }
            }

            $cont2 += 1;
        }
        if ($train_test < $num_ham_train){
            if ($tipoAnalisis eq "TF"){
            print TFTRAIN "\n";
            }
            else{
            print TFIDFTRAIN "\n";
            }
        }
        else{
          if ($tipoAnalisis eq "TF"){
            print TFTEST "\n";
          }
          else{
            print TFIDFTEST "\n";
          }
        }
        $train_test+=1;
}
$train_test=0;
imprimeHora("Se va a recorrer el fichero tfspam");
while (<TFSPAMLECTURA>) {

        #chop($_);
        my @tf_spam = split(/ /, $_ );
        my @solo_tf_spam = @tf_spam;
        my %hash_TF_IDF;
        my $term_temp;
        my $suma = 0;
```

```
        #imprimeHora("Calculando productos y suma");
        my $cont = 0;
        foreach $term_temp (@array_features) {
               my $producto;
               if ( $hash_DF{$term_temp} != 0 ) {

        $producto=$tf_spam[$cont]*log($num_mensajes/$hash_DF{$term_temp})/log(10
);
               }
               else {
                      $producto = 0;
               }
               $tf_spam[$cont] = $producto;
               $suma += ($producto)**2;
               $cont += 1;
        }
        my $raiz = sqrt($suma);
        my $term_temp2;
        my $cont2 = 0;
        if ($train_test < $num_spam_train){
               if ($tipoAnalisis eq "TF"){
              print TFTRAIN "+1 ";
               }
               else{
              print TFIDFTRAIN "+1 ";
               }
        }
        else{
               if ($tipoAnalisis eq "TF"){
              print TFTEST "+1 ";
               }
               else{
              print TFIDFTEST "+1 ";
               }
        }
```

```
	foreach $term_temp2 (@array_features) {
		if ( $raiz == 0 ) {
			$tf_spam[$cont2] = 0;
		}
		else {
			$tf_spam[$cont2] = $tf_spam[$cont2] / $raiz;
		}
		if ( $tf_spam[$cont2] != 0 ) {
			if ($train_test < $num_spam_train){
				if ($tipoAnalisis eq "TF"){
			print TFTRAIN $hash_features_orden{$term_temp2} . ":" .
$solo_tf_spam[$cont2] . " ";
				}
				else{
			print TFIDFTRAIN $hash_features_orden{$term_temp2} . ":" .
$tf_spam[$cont2] . " ";
				}
			}
			else{
				if ($tipoAnalisis eq "TF"){
			print TFTEST $hash_features_orden{$term_temp2} . ":" .
$solo_tf_spam[$cont2] . " ";
				}
				else{
			print TFIDFTEST $hash_features_orden{$term_temp2} . ":" .
$tf_spam[$cont2] . " ";
				}
			}
		}

		$cont2 += 1;
	}
	if ($train_test < $num_spam_train){
		if ($tipoAnalisis eq "TF"){
	      print TFTRAIN "\n";
		}
```

```
            else{
            print TFIDFTRAIN "\n";
             }
      }
      else{
            if ($tipoAnalisis eq "TF"){
            print TFTEST "\n";
             }
            else{
            print TFIDFTEST "\n";
             }
      }
      $train_test+=1;
}
print `date`;
sub sustituyeVocales {
my $var = $_[0];
$var =~ s/á/a/g;
$var =~ s/é/e/g;
$var =~ s/í/i/g;
$var =~ s/ó/o/g;
$var =~ s/ú/u/g;
$var =~ s/ü/u/g;
$var =~ s/Á/A/g;
$var =~ s/É/E/g;
$var =~ s/Í/i/g;
$var =~ s/Ó/O/g;
$var =~ s/Ú/U/g;
$var =~ s/Ü/U/g;
return $var;
}

sub imprimeHora {
my $texto = $_[0];
```

```
my $mensaje_log;

my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst);

($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);

$mensaje_log = ($mday."-".$mon."-".($year+1900)." ".$hour.":".$min.":".$sec."
".$texto."\n");

print $mensaje_log;

}
```

14 ANEXO D – CÓDIGO FUENTE PARA CUSTOMFUNCTIONS DE MAILSCANNER

```
#
#   MailScanner - SMTP E-Mail Virus Scanner
#   Copyright (C) 2002  Julian Field
#
#   $Id: GenericSpamScanner.pm,v 1.1.2.5 2005/07/12 11:27:14 jkf Exp $
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation; either version 2 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA
#
#   The author, Julian Field, can be contacted by email at
#      Jules@JulianField.net
#   or by paper mail at
#      Julian Field
#      Dept of Electronics & Computer Science
#      University of Southampton
#      Southampton
```

```
#       SO17 1BJ
#       United Kingdom
##########################################
#       CustomFunction desarrollada por Carlos Rubio Prieto
#       Ejecuta análisis antispam del correo electrónico y devuelve
#       su scoring a MailScanner.
#       La clasificación se realiza con una Máquina de Vectores Soporte
#       basada en el clasificador GSVC
##########################################

package MailScanner::CustomConfig;

use strict;
use IPC::Open2;
use FileHandle;
use Mail::MboxParser;
use HTML::Tagset;

sub GenericSpamScanner {
    my($ip, $from, $to, $message) = @_;

    my $numfeatures = MailScanner::Config::Value('numfeatures');
    my $tipoAnalisis = MailScanner::Config::Value('tipoAnalisis');

    MailScanner::Log::InfoLog("numfeatures = \"$numfeatures\"\n");
    MailScanner::Log::InfoLog("tipoAnalisis = \"$tipoAnalisis\"\n");
    MailScanner::Log::InfoLog("\n");
    MailScanner::Log::InfoLog("IP = \"$ip\"\n");
    MailScanner::Log::InfoLog("From = \"$from\"\n");
    MailScanner::Log::InfoLog("To = \"" . join(", ", @$to) . "\"\n");
    MailScanner::Log::InfoLog("Message = \"" . join(", ", @$message) .
"\"\n");
```

```
            if ($tipoAnalisis ne "TF" && $tipoAnalisis ne "TFIDF")

            {

                MailScanner::Log::InfoLog("Formato incorrecto\n");

                MailScanner::Log::InfoLog("NUMEROFEATURES debe ser un entero
positivo \n");

                #MailScanner::Log::InfoLog("y TIPOANALISIS puede tomar los
valores: TF, TFIDF\n");

                return (0.0, 'No report, error en GenericSpamScanner. Ver
maillog');

            }

            if ( $tipoAnalisis eq "TF")

            {

                open(".features100000_ord_TF.txt" ) || die "No se pudo abrir el
fichero de features100000_ord_TF\n";

                MailScanner::Log::InfoLog("se abre el fichero de
features100000_ord_TF.txt");

            }

            else

            {

                open(".features100000.txt" ) || die "No se pudo abrir el fichero
de features100000\n";

                MailScanner::Log::InfoLog("se abre el fichero de
features100000.txt");

            }


            my @array_features_t;

            my @array_features;

            while (<FEATURES>) {

                my $var = $_;

                @array_features_t = split(/ /,$var);

            }

            @array_features = @array_features_t[0..($numfeatures-1)];

            @array_features_t = [];
```

```
            my $tempo;

            my %hash_features_orden;

            my %hash_features;

            my $contador = 1;

            foreach $tempo (@array_features) {

                $hash_features_orden{$tempo} = $contador;

                $hash_features{$tempo}       = 0;

                $contador += 1;

            }

            my %hash_DF = %hash_features;

            open(STOPWORDS,".stopwords_all.txt") || die "No se pudo abrir el
fichero de stopwords\n";

            my %stopwords;

            while (<STOPWORDS>) {

                chop($_);

                $stopwords{$_} = 1;

            }

            MailScanner::Log::InfoLog("Se han leido las stopwords");

            open(HASHDF,".hashDF.txt") || die "No se pudo abrir el fichero
hashDF.txt\n";

            my $palabraEnHashDF;

            my $DFdePalabra;

            while (<HASHDF>) {

                ($palabraEnHashDF,$DFdePalabra) = split($_);

                $hash_DF{$palabraEnHashDF} = $DFdePalabra;

            }

            open(NUMMENSAJES,".numeroMensajes.txt") || die "No se pudo abrir el
fichero numeroMensajes.txt\n";

            my $num_mensajes;

            while (<NUMMENSAJES>) {

                chop($_);

                $num_mensajes = $_;

            }
```

```
            $num_mensajes += 1;


            my $cuerpo = $message->body( $message->find_body );
        my $texto = $cuerpo->as_string( [ strip_sig => 1 ] );
        my $longitudtexto=length($texto);
            if ($longitudtexto>=1000000)
            {

MailScanner::Log::InfoLog("longitudtexto:\".length($texto).\"\n");
                #MailScanner::Log::InfoLog("El correo es demasiado largo. No se
analiza\n");
            }
            ## Quito acentos y dieresis ##
            $texto = &sustituyeVocales($texto);
            ## Paso a minusculas ##
        $texto = lc $texto;
        my @palabras_correo;
        @palabras_correo = split( /\s+|\W+/, $texto );
        my $palabra;
        my %hash_local;
        print ("1: recorro palabras correo");
        foreach $palabra (@palabras_correo) {
                if ( ( exists $stopwords{$palabra} ) || ( $palabra eq "" ) ) {
                    next;
                }
               else {
                    if ( ( exists $hash_features{$palabra} ) ) {
                        if ( exists $hash_local{$palabra} ) {
                            $hash_local{$palabra} += 1;
                            print ($palabra);
                        }
                        else {
```

```
                        $hash_local{$palabra} = 1;
                        print ($palabra);
                    }
                }
                else {
                    next;
                }
            }
        }
    my $termino;
    foreach $termino ( keys %hash_local ) {
            $hash_DF{$termino} += 1;
        }

        my %hash_operaciones;
        my $term_temp;
        my $suma = 0;
        foreach $term_temp (@array_features) {
            my $producto;
            if ( $hash_DF{$term_temp} != 0 ) {
                if ( exists $hash_local{$term_temp}) {
                    $producto=$hash_local{$term_temp}*log(
$num_mensajes/$hash_DF{$term_temp})/log(10);
                }
                else {
                    $producto = 0;
                }
            }
            else {
                    $producto = 0;
            }
                $hash_operaciones{$term_temp} = $producto;
```

```
                print("\n2.5:producto:".$producto);
                $suma += ($producto)**2;
                print("\n2.6:suma:".$suma);
        }
        my $raiz = sqrt($suma);
        print("\n3:raiz:".$raiz);
        my $term_temp2;
        open(TFIDF,">tfidf.txt");
        print TFIDF "+1 ";
    foreach $term_temp2 (@array_features) {
            if ( $raiz == 0 ) {
                $hash_operaciones{$term_temp2} = 0;
            }
            else {
                $hash_operaciones{$term_temp2} = $hash_operaciones{$term_temp2} / $raiz;

            }
            if ( $hash_operaciones{$term_temp2} != 0 ) {
                print TFIDF $hash_features_orden{$term_temp2} . ":" . $hash_operaciones{$term_temp2} . " ";
             }
    }
        print TFIDF "\n";
        close(TFIDF);
        unless(fork){
            exec "./gsvc_classify -f tfidf.txt | grep ACIERTO > ACIERTO.txt";
        }
        wait;
        open(ACIERTO,"ACIERTO.txt");
        my @resultado;
        while(<ACIERTO>){
```

```
        my $cadena=$_;
        chop($cadena);
        print $cadena;
        @resultado = split(/\s+/,$cadena);
    }
    my $scoring=$resultado[4];
    print "\nScoring ".$scoring;
    $scoring=$scoring*0.2;
print "\nScoring ".$scoring;
unlink("tfidf.txt","output.txt","gmon.out","ACIERTO.txt");
return ($scoring, 'Scoring obtenido con GenericSpamScanner');
}

sub sustituyeVocales {
my $var = $_[0];
$var =~ s/á/a/g;
$var =~ s/é/e/g;
$var =~ s/í/i/g;
$var =~ s/ó/o/g;
$var =~ s/ú/u/g;
$var =~ s/ü/u/g;
$var =~ s/Á/A/g;
$var =~ s/É/E/g;
$var =~ s/Í/i/g;
$var =~ s/Ó/O/g;
$var =~ s/Ú/U/g;
$var =~ s/Ü/U/g;
return $var;
}

1;
```

15 ANEXO E – CÓDIGO FUENTE PARA EL ENVÍO AUTOMÁTICO DE CORREOS A TRAVÉS DE SENDMAIL

```
use strict;

use Mail::MboxParser;

use HTML::Tagset;


print `date`;

my $num_mensajes = 0;

my $num_ham = 0;

my $num_spam = 0;

my $mbSpamTrain = Mail::MboxParser->new( '../caso5/buzones_spam/TRAIN',
decode => 'ALL' );

my $mbHamTrain = Mail::MboxParser->new( '../caso5/buzones_ham/TRAIN',
decode => 'ALL' );

my $accion=0;

print "Introduzca accion:\n";

print "0: Terminar\n";

print "1: Enviar ham\n";

print "2: Enviar spam\n";

$accion=<STDIN>;

chop($accion);

BUCLE: while ($accion != 0){

    my $msg;

    if ($accion == 1) {

        $msg = $mbHamTrain->next_message;

        print "Se lee correo ham\n";

    }

    else {

        if ($accion == 2){

            $msg = $mbSpamTrain->next_message;
```

```
            print "Se lee correo spam\n";
        }
        else {
            print "Accion incorrecta. Introduzca 0, 1 ó 2\n";
            $accion=<STDIN>;
            chop($accion);
            next BUCLE;
        }
    }
    my $cuerpo = $msg->body( $msg->find_body );
    my $texto = $cuerpo->as_string( [ strip_sig => 1 ] );
    my $longitudtexto=length($texto);
    if ($longitudtexto>=1000000)
    {
        next BUCLE;
    }

    my $sendmail = "/usr/sbin/sendmail -t";
    my $reply_to = "Reply-to: root\@localhost.localdomain\n";
    my $subject  = "Subject: Prueba de deteccion de correo basura.Opcion
".$accion."\n";
    my $content  = $texto;
    my $to       = "To: carlos\@localhost.localdomain\n";

    print "Se va a enviar correo \n";

    open(SENDMAIL, "|$sendmail") or die "Cannot open $sendmail: $!";
    print SENDMAIL $reply_to;
    print SENDMAIL $subject;
    print SENDMAIL $to;
    print SENDMAIL "Content-type: text/plain\n\n";
    print SENDMAIL $content;
```

```
    close(SENDMAIL);


    print "Correo enviado\n";
    print "Introduzca accion:\n";
    print "0: Terminar\n";
    print "1: Enviar ham\n";
    print "2: Enviar spam\n";
    $accion=<STDIN>;
    chop($accion);
}


print "Finalizado\n";
```