Detección de correo basura mediante técnicas de Inteligencia Artificial

por
Alberto Martínez de Murga Ramírez
DNI: [redacted]
alberto.mmr@gmail.com

Indice

Descripción del problema.

El problema a tratar por el filtrado de correo basura. El filtrado de correo basura viene a consistir en, una vez recibido un correo, saber distinguir si se trata de un correo cuyo contenido es legítimo, es decir, *ham*; o si por el contrario, se trata de un correo no deseado, habitualmente publicitario, conocido como *spam*.

Se trata por tanto de un problema de clasificación de documentos, es decir, decidir en base a ciertas características y propiedades a qué clase pertenece el documento. Es un problema típico abordable desde la Inteligencia Artificial por medio de diversos algoritmos de clasificación como algoritmos bayesianos o sistemas de razonamiento.

Para este caso concreto expuesto es necesario que, además de resolver el problema en cuestión, se cumplan una serie de condiciones:

- Se deberá de utilizar como corpus de referencia y evaluación el corpus *Enron*, disponible en la siguiente dirección: http://www.aueb.gr/users/ion/data/enron-spam/
- El sistema deberá incluir tres módulos operativos: uno de entrenamiento, que deberá de ser capaz de a partir de una lista de correos generar un clasificador y almacenarlo; uno de evaluación, que dado un modulo de aprendizaje previamente almacenado y uno de correos, deberá de clasificarlos y obtener un estudio estadístico de la clasificación; y un módulo de clasificación, el cual dado un modulo de entrenamiento y un correo deberá decidir si se trata de spam o no.
- Se deberá de incluir los resultados de dos evaluaciones, la primera usando *Enron* de 1 a 5 de como entrenamiento y 6 como evaluación, y la segunda que a partir de el conjunto de todos los *Enron*, empleará un 75% de los correos como entrenamiento y el 25% para evaluación.

Análisis y diseño del algoritmo.

El algoritmo en cuestión que se ha implementado es una versión del algoritmo k-NN. El algoritmo k-NN es un clasificador típico que, para clasificar un documento, busca en el conjunto de entrenamiento los *k* documentos más cercanos y devuelve la clase mayoritaria como clasificación. La cercanía se puede medir de muchas formas, pero por lo general se suele recurrir a una medida vectorial. Resulta de especial importancia encontrar un valor de *k* que permita una correcta clasificación. No existe un método empírico para calcularlo, siendo habitualmente el hacer pruebas para pequeños conjuntos la forma de averiguarlo.

Este sistema no suele ser muy usado para la clasificación de spam, sin embargo, existen implementaciones exitosas del mismo, como es el sistema *ECUE*, del cual se puede encontrar información en el siguiente enlace: http://arrow.dit.ie/scschcomcon/24/

En un principio, el enfoque original del algoritmo a emplear en el proyecto era una simple implementación del algoritmo ECUE, sin embargo, este emplea más de 700 características para calcular las distancias entre los emails de las que no habla en ningún momento.

Por tanto, tomé la decisión de realizar una aproximación de la abstracción del conjunto de emails en forma de espacio de vectores, que es otra técnica de abstracción propia de la inteligencia artificial.

Un espacio vectorial mide la relevancia de un documento en función de la frecuencia con la que aparecen las palabras en el documento y el conjunto de todos los documentos. A más apariciones, más relevante es la palabra, pero menor si esta aparece en muchos documentos.

Para obtener un espacio de vectores, generamos una tabla cruzando las palabras que consideramos relevantes del conjunto de los documentos con los documentos junto con los siguientes campos:

- **Term Frequency** ($tf_{t,d}$): Es el número de veces que aparece dicho término en dicho documento.
- **Documental Frequency** (df_t): Es el número de documentos en los que aparece el término.
- I**nversal Documental Frequency** (idf_t): Es el logaritmo del numero total de documentos partido del df_t de un término.

A partir de esta tabla, se puede generar para un documento un vector de pesos, el cual calcula para cada palabra del vocabulario en el documento su peso, multiplicado el peso

de dicha palabra por el td_f de la palabra, siendo 0 para aquellas palabras que no aparecen en el vocabulario.

Asociando dentro de un espacio de vectores a un documento un valor vectorial numérico podemos calcular la similitud de los dos documentos como medida de distancia, tal que cuanto mayor sea la distancia, menor será la similitud. La similitud sigue la siguiente formula:

$$sim(\vec{V}, \vec{W}) = \frac{\sum_{i=1} V_i W_i}{\sqrt{\sum_{i=1}(V_i)^2}\sqrt{\sum_{i=1}(W_i)^2}}$$

Por tanto, se consigue de esta forma una manera de calcular la distancia para el funcionamiento del algoritmo k-NN.

La secuencia del algoritmo sería la siguiente:

i. Para cada correo del corpus de entrenamiento:
   (a) Eliminar las stop words (palabras carentes de valor semántico) y signos de puntuación.
   (b) Obtención del tf de cada palabra del documento y asociarlo en una estructura a cada palabra.
ii. Almacenar el vocabulario en un conjunto generado a partir de la unión de todas las palabras no repetidas del documento.
iii. Calcular el espacio de vectores asociado al corpus de entrenamiento.
iv. Generar el vector asociado a cada correo del conjunto de entrenamiento en base al espacio de vectores.
v. Seleccionar aleatoriamente y almacenar en un conjunto un porcentaje no mayor de 1000 vectores de correos, mezclando al 50% ham y spam.
vi. Para cada correo del corpus de entrenamiento
   (a) Clasificar por medio de k-NN el vector asociado al correo, buscando los k correos más cercanos en el conjunto de clasificación al correo a clasificar e asignándole la clase mayoritaria.
   (b) Si la clasificación no coincide con la del correo, añadir al conjunto de clasificación siempre que no supere el conjunto de clasificación el 25% del corpus.
vii. Emplear el conjunto de clasificación como clasificador. Para clasificar un nuevo correo, solo hay que calcular su vector asociado y obtener su clasificación k-NN.

El diseño del algoritmo, y sobre todo, el entrenamiento están basados en el funcionamiento de *ECUE*. Parte de reducir el conjunto de comparación a un mínimo para ir ampliándolo durante el entrenamiento en función de las necesidades de la clasificación. La idea es que si no ha sido clasificado correctamente por el conjunto clasificación es porque aporta información relevante al conjunto y debe de ser añadido.

La limitación a la cantidad de correos del conjunto es una medida contra el sobreentrenamiento, y para evitar aumentar el tiempo de ejecución del sistema, puesto que debe de revisar todos los correos almacenados en el conjunto de entrenamiento. En un entorno de producción, las cifras podrían ser mayores, debido a la mayor cantidad de recursos disponibles.

Diferentes pruebas realizadas durante el diseño del algoritmo corroboraron que dejar crecer indefinidamente el conjunto de entrenamiento no mejoraba significativamente los resultados, pero si aumentaban significativamente los costes de memoria y tiempo tanto del entrenamiento como de la evaluación.

Implementación.

La implementación del algoritmo se ha realizado en el lenguaje de programación Java. Se ha dividido su diseño en clases siguiendo los preceptos de la orientación a objetos. Se ha empleado además la librería *Guava* (http://code.google.com/p/guava-libraries/) con el fin de facilitar el tratamiento de las grandes colecciones de datos, proporcionando funcionalidades adicionales a las clases de Java. El código fuente se ha adjuntado comentado para su mejor comprensión. La clase **Main** del paquete **Main** es el punto de acceso al programa, y siguiendo su ejecución se puede ver el planteamiento lógico que sigue el sistema.

Instrucciones de instalación y uso.

El sistema se distribuye en forma de ejecutable jar que no requiere instalación adicional alguna a la de Java en el sistema operativo. Es una aplicación de consola la cual solo hay que ejecutarla desde la linea de comandos. Estando situados en el mismo directorio, solo hay que ejecutar:

```
java -jar spamfilter.jar
```

Posteriormente, solo hay que seguir las instrucciones que aparecen en pantalla. No obstante, es importante tener en cuenta una serie de consideraciones, las cuales han sido expuestas además al inicio del programa.

- Siga las instrucciones que aparecen en pantalla.
- Incluya la ruta completa a los archivos, prestando atención a la extensión.
- Use en todo momento tanto para clasificación como para evaluación archivos del conjunto Enron.
- Evite emplear espacios en las rutas.

Resultados del entrenamiento y la evaluación.

Los resultados anotados para los entrenamientos son el peso final del archivo donde se almacena, la cantidad de emails procesados y el tiempo empleado. Para la evaluación, es el tamaño del corpus, del vocabulario, del conjunto de evaluación y el tiempo empleado en la evaluación, junto con los resultados obtenidos.

Todas las pruebas se han realizado sobre un Macbook corriendo **OS X 10.8.4** con un procesador **Intel Core 2 Duo 2.13 GHz**, **4GB 800MHz DDR2 SDRAM** y usando **Java SE 7**.

Evaluación 1: Utilizando Enron 1 a 5 como entrenamiento y Enron 6 como evaluación.

Entrenamiento

- Conjunto de entrenamiento: 27716 emails.
- Tiempo de entrenamiento: 16 minutos y 18.85 segundos.
- Tamaño del archivo: 6,1 MB.

Evaluación

- Tamaño del conjunto de evaluación: 1121 emails.
- Tamaño del vocabulario: 136101 palabras.
- Tamaño del corpus de evaluación: 6000 emails.
- Tiempo empleado: 29 minutos y 44.54 segundos (0.29 segundos por email).
- Resultados:
    - Total de mensajes de spam: 4500
    - Total de mensajes de ham: 1500
    - Mensajes de spam incorrectamente clasificados: 353 (7.84%)
    - Mensajes de ham incorrectamente clasificados: 146 (9.73%)

Evaluación 2: Partiendo de una división (aleatoria) de los mensajes de Enron 1 a Enron 6 en dos bloques (de entrenamiento con el 75% y de evaluación con el 25%).

Entrenamiento

- Conjunto de entrenamiento: 25287 emails.
- Tiempo de entrenamiento: 18 minutos y 32.33 segundos.
- Tamaño del archivo: 5,5 MB.

Evaluación

- Tamaño del conjunto de evaluación: 1218 emails.
- Tamaño del vocabulario: 121442 palabras.
- Tamaño del corpus de evaluación: 8429 emails.
- Tiempo empleado: 45 minutos y 20.17 segundos (0.32 segundos por email).
- Resultados:
    - Total de mensajes de spam: 4182
    - Total de mensajes de ham: 4247
    - Mensajes de spam incorrectamente clasificados: 394 (4.31%)
    - Mensajes de ham incorrectamente clasificados: 183 (9.42%)

Análisis crítico y posibles mejoras.

Tal y como se puede apreciar en el siguiente artículo http://polar.lsi.uned.es/revista/index.php/ia/article/viewFile/533/517, no existen muchos clasificadores basados en algoritmos k-NN, pero queda demostrado que es posible realizar un clasificador de spam usando como base este tipo de algoritmo. Su mayor problema no obstante es tener que enfrentarse a grandes conjuntos de evaluación y/o vocabulario, aumentando los tiempos de ejecución.

No obstante, *ECUE* y los mismos resultados aquí obtenidos demuestran que acotando el tamaño del vocabulario y del conjunto de evaluación se pueden obtener resultados aceptables. Una posible mejora al sistema actual sin duda sería reducir el tamaño del vocabulario con una gestión y filtrado más exhaustivo de las palabras que forman parte de él, prestando atención a posibles optimizaciones. Es importante resaltar que los resultados de este algoritmo son similares a los de *ECUE* sin actualización.

Otra característica que se ha implementado parcialmente pero no usado es la actualización del conjunto de evaluación y vocabulario en tiempo de ejecución. ECUE hace uso de esta característica demostrando en sus resultados una clara mejoría. Esta mejora consiste en, durante la ejecución, actualizar el conjunto de evaluación con los emails incorrectamente clasificados, junto con la actualización del vocabulario, siendo este uno de sus principales puntos fuertes.

Esta característica no se ha implementado debido a varias razones. En primer lugar, implica interacción por parte del usuario para que al detectar una clasificación incorrecta active el mecanismo. En segundo lugar, alargaría mucho el proceso de evaluación de emails en masa, siendo una característica más enfocada a la recepción habitual de emails que a pruebas en masa. En tercer lugar, implicaría actualizar por cada email añadido el espacio de vectores con el nuevo documento en si, por lo que implicaría también que realizar optimizaciones para evitar tener que recalcular el espacio de vectores al completo.

No obstante, se ha añadido código que no se ha empleado luego en la implementación final donde se muestra como de sencillo sería implementar una versión no optimizada de esta mejora.