

Filtros Bayesianos en Python

# EL SPAM SE VA A ACABAR

Paul Graham publicó un artículo en 2002 en el que decía que los filtros bayesianos acabarían con el Spam, pero ¿qué es un filtro bayesiano? y, peor aún ¿qué es en realidad el spam?

**POR JOSE MARÍA RUÍZ AGUILERA**

Es una buena pregunta y su contestación no es sencilla. Si definimos Spam como correo no solicitado, entonces no habría forma de determinar si un correo es spam o no. Pero afortunadamente el spam posee unas características bien definidas. Su mensaje es repetitivo, y el texto, tarde o temprano, tiene que hacer referencia a lo que nos intenta vender.

En 1998, M. Sahami, S. Dumais, D. Heckerman y E. Horvitz, cuatro estudiosos de la inteligencia artificial, se dieron cuenta del problema que se nos venía encima. Presentaron su trabajo *A Bayesian approach to filtering junk e-mail* (una aproximación bayesiana al filtrado del correo basura) en un congreso de la Asociación Americana de Inteligencia Artificial. A pesar de presentar unos resultados muy esperanzadores para la lucha contra el spam, esta técnica permaneció en la penumbra. Hasta que el spam se volvió insoportable.

En el año 2002, Paul Graham (ver Recurso [1]) escribió un ensayo titulado *A plan for Spam*, en el cual describía cómo programar un filtro bayesiano de spam. Paul estaba utilizando ese filtro desde hacía meses con gran éxito (según cuenta, con un 99.5 % de aciertos y ningún falso positivo).

Por cierto, la palabra spam se la debemos al genial grupo humorístico Monty Python, de cuyo nombre surgió el nombre Python. Se puede ver el vídeo original del que surgió el nombre spam en el Recurso [2].

En su ensayo Paul Graham cita a Norbert Wiener, padre de la cibernética, cuando dice «...si compites con esclavos acabas convirtiéndote en un esclavo». Y en eso se había convertido, en un esclavo de los spammers. Eran ellos quienes tenían las de ganar, porque son ellos quienes fijan las reglas del juego.

## La cruzada contra el spam

Después de unos meses trasteando con técnicas simples, como por ejemplo clasificar los correos en base a frases o palabras clave del estilo xanax sex, se dio cuenta de que sería una lucha sin fin. Por ejemplo, los spammers dejaron de usar la palabra «viagra» y comenzaron a usar «v1agra» e incluso «vagra».

Decidido a ganarles, pasó a investigar formas más sofisticadas de clasificación de textos, llegando hasta los filtros bayesianos. Implementó un nuevo filtro basado en ellos y el éxito fue extraordinario. Rápidamente todo el software de filtrado de correos incorporó una opción para emplear filtrado bayesiano.

Imagina que llegas a una parada del autobús urbano y ves que está vacía, ¿tardará mucho en llegar el siguiente autobús?

## Un poco de probabilidad

El cálculo de probabilidades parece una técnica difícil de entender, pero la usamos de forma inconsciente en el día a día. Por ejemplo, siendo usuario habitual del transporte urbano, debo tomar ciertas decisiones dependiendo de determinadas variables, y así tengo en cuenta:

- las distintas combinaciones de autobuses que me llevan a un mismo sitio
- la cantidad de tiempo que debo esperar a que llegue el autobús
- lo que tarda cada combinación de autobuses en llegar a su destino
- ...

Si es domingo por la tarde y la parada está vacía, no podremos realizar ninguna predicción, el transporte urbano no es muy usado en días festivos. En cambio, si es lunes por la mañana, en hora punta y la parada está vacía, podremos suponer que el autobús ha pasado recientemente.

En función de la información que está en nuestra mente tomamos decisiones, así

### Listado 1: Ejemplo de uso de conjuntos

```
01 >>> a = {}
02 >>> a['hola']=1
03 >>> a['adios']=2
04 >>> b = {}
05 >>> b['hola'] = 3
06 >>> b['coche'] = 4
07 >>> import sets
08 >>> c = sets.Set(a.keys()) |
    sets.Set(b.keys())
09 >>> l = []
10 >>> for palabra in c:
11 ...     l.append(palabra)
12 ...
13 >>> l
14 ['hola', 'adios', 'coche']
```

como con aquella otra que podemos extraer del entorno. Calculamos las probabilidades de forma aproximada en busca de la mejor opción. Si fuésemos capaces de crear un programa que tomase decisiones como lo hace nuestra mente, habríamos sido los afortunados programadores creadores de la primera inteligencia artificial.

Nuestro programa sólo puede operar con las matemáticas, así que, ¿cómo se pueden modelar estas decisiones empleando matemáticas?

## Bayes

El teorema de Bayes (o la probabilidad condicionada) es un ejemplo de matemática útil y práctica. Nos dice que la probabilidad de tomar el autobús sabiendo que es domingo es igual a la probabilidad de que siendo domingo se coja el autobús, por la probabilidad de coger el autobús y dividido por la probabilidad de que sea domingo.

Volviendo al tema del filtrado de correos, ¿cómo sería esta fórmula en nuestro contexto? La probabilidad de que un correo sea spam sabiendo que contiene ciertas palabras («hola», «venta», «nigeria», ...) es igual a la probabilidad de que esas palabras estén en un correo spam («nigeria» es una palabra típica del spam) por la probabilidad de que un correo cualquiera sea spam (número de correos spam entre total de correos) dividido por la probabilidad de que aparezcan esas palabras en un correo cualquiera.

La palabra «probabilidad» ha aparecido tantas veces, que estoy seguro que el lector se habrá asustado. Contrariamente a lo que pueda creer, la probabilidad es lo más sencillo de calcular en todo este asunto. Simplemente hay que realizar una división: la probabilidad de que una palabra aparezca en spam es igual al número de veces que aparece esa palabra en nuestro spam, dividido por el número de veces que aparece en nuestros correos, sean spam o no.

Digamos que tenemos 100 correos, de los que 35 son spam y el resto, 65, no lo son. La palabra nigeria aparece 8 veces, 6 en correos spam y 2 en correos normales. La probabilidad es un número entre 0 y 1. En este caso, un 0.847, representa un 84% de probabilidad. Por tanto un correo con la palabra nigeria es muy probablemente spam.

## Los filtros bayesianos

Los filtros bayesianos han conseguido gran popularidad. En la Tabla [1] se pueden ver algunos ejemplos de uso real de esta técnica.

En nuestra implementación del filtro bayesiano vamos a necesitar alguna estructura de datos para guardar las apariciones de cada palabra tanto en el correo normal como en el spam.

Emplearemos diccionarios. Usaremos dos directorios. En uno, al que llamaremos «spam», guardaremos ficheros de texto con el cuerpo de correos spam. En el otro directorio, llamémosle «ok», guardaremos los ficheros con el cuerpo de correos normales.

Nuestro primer paso consistirá en extraer las palabras que aparecen en los correos de ambos tipos y contar el número de apariciones de esas palabras. Entonces las guardaremos en dos diccionarios. Así si `ok["nigeria"]` es igual a 2, significará que la palabra «nigeria» aparece dos veces en nuestros correos normales, y si `spam["nigeria"]` es 5, nos vendrá a decir que nigeria aparece 5 veces en nuestros correos spam.

Este proceso es rutinario:

- abro fichero
- genero una lista de sus palabras
- guardo o actualizo su entrada en el diccionario correspondiente

En palabras llanas, estamos creando un inventario de palabras.

Para simplificar el código sólo romperemos el texto de forma simple, aprovechando la función `split()` que Python incorpora en sus cadenas. Esta función divide la cadena usando separadores estándar como pueden ser el punto o el espacio en blanco.

Con nuestros dos diccionarios cargados de palabras, pasamos a la segunda fase: la creación de un diccionario conjunto donde guardemos la probabilidad de cada palabra de ser parte de un correo spam.

Este tercer diccionario debe contener las palabras del diccionario `ok` del diccionario `spam`. Para ello debemos crear un diccionario que incorpore las entradas de ambos, pero una sola vez para cada uno, ¿cómo podemos mezclar las llaves de ambos diccionarios?

La forma más sencilla es la que aparece en el Listado [1]. Importamos la librería `sets` que nos permite crear y manejar conjuntos. Creamos dos conjuntos usando las llaves de los diccionarios y los unimos con el operador `|`, la unión de conjuntos, que devuelve un conjunto con las entradas de los conjuntos unidos, pero sin repeticiones. Los conjuntos pueden tener entradas repetidas, pero cuando recorremos el conjunto con `for` no aparecen, porque los conjuntos sólo pueden devolver un elemento de cada tipo aunque estén repetidos. De esta forma tan sencilla

podemos obtener una lista con las llaves de ambos diccionarios.

El siguiente paso es extraer las llaves de esa lista de llaves compartidas, guardando las entradas en una lista, y calcular la probabilidad de ser spam para cada una de ellas con una fórmula que pasamos a explicar.

## Falsos positivos

Uno de los problemas que encontramos cuando clasificamos algo es la aparición de falsos positivos.

Esta expresión a veces suele escucharse en ámbitos médicos. Indica que se ha detectado algo, pero que en realidad no hay nada debido a que la prueba ha fallado.

Por ejemplo, estamos enfrente de la parada del autobús, y vemos que tendremos dificultades para tomarlo, ya que un semáforo nos interrumpe el paso, por lo que lo damos por perdido.

Pero conforme el semáforo se pone en verde el autobús sigue ahí, y como le dimos por perdido atravesamos el paso de cebra a paso normal. Sólo cuando estamos a mitad de él y aún vemos el autobús nos damos cuenta de nuestro fallo de razonamiento: si hubiésemos salido corriendo por el paso de cebra al ponerse el semáforo en verde podríamos haber tomado ese autobús. Después de este falso positivo se nos suele quedar cara de tontos, mientras el autobús se va sin nosotros a sólo unos pasos de distancia.

Lo mismo ocurre con los filtros antispam. Si confiamos completamente en ellos, un día puede ocurrir que aparezca un falso positivo, que sea clasificado como spam y que nunca lleguemos a enterarnos de la llegada de un correo importante.

Tenemos que minimizar la aparición de falsos positivos tanto como sea. Para ello tendremos que asumir que es preferible que

### Listado 2: función `es_spam()`

```
01 >>> def es_spam(probs):
02 ... prod=reduce(lambda x, y: x
03 ... * y, probs)
04 ... return prod / (prod +
05 ... reduce(lambda x,y: x*y,
06 ... map(lambda x: 1 - x, probs)))
07 >>> f(p)
08 0.034848854359010119
```

se cuele un poco de spam en nuestra bandeja de entrada a perder un correo importante. Es por esto que vamos a trabajar con una fórmula algo diferente a la de Bayes, emplearemos pesos. Un peso es un número que multiplica una cantidad para hacerla más grande, y por tanto, para hacer que «pese» más dentro de la fórmula.

## La fórmula

La fórmula, que podemos ver en el Listado 2, hace uso de las funciones *max()* y *min()*, que devuelven el máximo y el mínimo de sus argumentos. El objetivo de su uso es no permitir la aparición de valores ni demasiado pequeños ni demasiado grandes. Cualquier valor mayor que 0.99 pasará a ser 0.99, y cualquier valor menor que 0.1 se convertirá en 0.1. Esta técnica nos permite establecer un umbral con el que podremos controlar el acierto de nuestro filtro.

Otro paso para mejorar la fórmula es no emplear la suma de correos correctos y spam. En su lugar usaremos sólo la cantidad de correos correctos o spam para calcular las probabilidades correspondientes. El objetivo de esto es ver lo importante que es una palabra dentro de uno de los grupos de correos, *oky spam*, y no en base al total.

Es momento de ver la fórmula en acción, ver Listado [\*]. La función acepta un solo parámetro, la palabra de la que queremos calcular la probabilidad. Los valores *g* y *b* (de good y bad) guardan la cantidad de apariciones de la palabra en el correo normal y el spam respectivamente. Como queremos tener tan pocos falsos positivos como sea posible, multiplicamos por 2 la cantidad de apariciones de las palabras en el correo normal. De esta forma damos más peso a las palabras que aparecen en el correo normal.

Tenemos cuidado de comprobar primero si la palabra se encuentra en ese diccionario. El problema surge cuando la palabra no se encuentra en ninguno de los dos diccionarios, porque en tal caso, en la fórmula aparece una división por 0. Un número dividido por 0 generará un error, debido a que la división por 0 no está definida en matemáticas.

¿Y qué hacemos si se da este caso? Pues como convenio devolveremos una probabilidad de 0.0, y por eso guardamos ese valor en el variable *resultado*.

## Implementación

Vista la teoría vayamos a la práctica. Nuestro programa, ver Listado 3, se compone de dos clases. La primera, *correos*, sirve como una ayuda para la segunda *filtro\_bayesiano*. En esta segunda clase es donde está lo más importante.

El método *\_\_init\_\_* de *filtro\_bayesiano* muestra los pasos necesarios para inicializar el filtro que antes comentamos:

- cargamos las palabras de los correos en diccionarios
- generamos el diccionario con las probabilidades

Para ello nos ayudamos del método *analiza\_correos*. Usando la librería *glob*, que nos permite conseguir listados de ficheros, creamos una lista con los ficheros en el directorio de correo normal o spam, según el caso. Recorremos esa lista de ficheros, abriéndolos y pasando el texto que contienen a *carga\_correo*. Como último paso, aumentamos en 1 el número de correos leídos de cada tipo, que nos permitirá posteriormente realizar el cálculo de probabilidades.

*carga\_correo* rompe el texto en palabras, y si la palabra no estaba en el diccionario correspondiente, que pasamos como parámetro, crea la entrada inicializándola a 1. En caso de que la palabra se hallase en el diccionario, aumentamos su cantidad en 1.

Con lo realizado hasta el momento tendremos listos los dos diccionarios, uno para correo normal y otro para spam. Ahora toca aplicar la fórmula.

## Diccionario de probabilidades

El método *construye\_diccionario* es el encargado de extraer las llaves, las palabras, de los dos diccionarios y almacenar en un tercero la probabilidad de cada palabra mediante el método *calcula\_prob\_spam*.

El código de *calcula\_prob\_spam* tiene en cuenta la posible aparición de una división por cero, por lo que comprueba que el divisor, el número que divide, no sea cero con una simple suma ( $(g + b) > 0$ ). Devuelve un resultado numérico entre 0,0 y 1,0, que será la probabilidad de que esa palabra pertenezca a un correo con spam.

Ya está listo nuestro filtro para decidir si un texto es spam o no.

## ¿Es esto spam?

Para tomar la decisión, troceamos el texto que se nos pasa como argumento, con lo que obtenemos una lista de palabras, y realizamos las siguientes acciones:

- Mapeamos sobre la lista el método *prob\_spam\_palabra* (ver Listado 3), lo que nos devuelve una lista de probabilidades.
- Multiplicamos esas probabilidades unas con otras hasta tener un solo número.
- Dividimos ese valor, que llamamos *prod*, por ese mismo valor más el resultado de multiplicar las probabilidades de que esas palabras no sean spam (este valor lo conseguimos restando esas probabilidades a 1)

El resultado es un valor numérico entre 0.0 y 1.0. Cuanto mayor sea, más probable es que estemos analizando un correo spam. Como regla general, consideraremos spam cualquier valor por encima de 0.9, de forma que los falsos positivos sean mínimos.

## Conclusión

Cuando se lee acerca de técnicas matemáticas empleadas para resolver grandes problemas, normalmente pensamos que no seremos capaces de comprenderlas. Nada más lejos de la realidad.

La matemática es una herramienta muy poderosa y aparece constantemente en nuestras vidas. Incluso con unos conocimientos básicos de pensamiento matemático, y no tanto de cálculo numérico como nos suelen enseñar, podemos dar soluciones simples y elegantes a problemas realmente complicados. ■

**Tabla 1: Filtros anti-spam con filtro bayesiano**

Nombre	Lenguaje	URL
Spambayes	Python	<a href="http://spambayes.sf.net">http://spambayes.sf.net</a>
POPFile	Perl	<a href="http://popfile.sf.net">http://popfile.sf.net</a>
SpamTUNNEL	Java	<a href="http://uiorean.cluj.astral.ro">http://uiorean.cluj.astral.ro</a>
PASP	Python	<a href="http://sf.net/projects/pasp">http://sf.net/projects/pasp</a>
Thunderbird	C++	<a href="http://www.mozilla.org/mailnews/spam.html">http://www.mozilla.org/mailnews/spam.html</a>
SpamProb	Python	<a href="http://spamprobe.sourceforge.net">http://spamprobe.sourceforge.net</a>

## RECURSOS

- [1] Solución de Paul Graham al problema del Spam: <http://www.paulgraham.com/spam.html>
- [2] Monty Python definen el Spam en su serie de televisión "Monty Python's Flying Circus": <http://www.youtube.com/watch?v=d7uFntk0Bnk>

## Listado 3: Nuestro filtro

```

001 #!/usr/local/bin/python
002 # # -*- coding: utf-8 -*-
003
004 import glob, os, sys, sets
005
006 class correos:
007     # Esta clase es para ahorrar
    código
008     def
    __init__(this,nombre,ruta):
009         this.ruta=ruta
010         this.nombre = nombre
011         this.num = 0.0
012         this.palabras = {}
013
014     def __str__(this):
015         return "[Correos
    "+this.nombre+"] cantidad:
    "+str(this.num)
016
017 class filtro_bayesiano:
018
019     def __init__(this):
020         this.ok =
    correos('ok','./ok/')
021         this.spam =
    correos('spam','./spam/')
022         this.probs_spam = {}
023         this.carga()
024         this.construye_diccionario()
025
026     def carga(this):
027
    this.analiza_correos(this.ok)
028
    this.analiza_correos(this.spam
    )
029
030     def
    analiza_correos(this,correos):
031         # Recibe una clase correos y
    usa sus datos
032         # para cargar en ella las
    palabras
033
034         lista =
    glob.glob(correos.ruta+"/*.txt
    ")
035         for f in lista:
036             fichero = open(f)
037
    this.carga_correo(correos,fich
    ero.read())
038         correos.num += 1.0
039
040         # sacamos las llaves de ambos
    diccionarios, y las mezclamos
    en un
041         # conjunto del que vamos
    extrayendo.
042
043     def
    calcula_prob_spam(this,palabra
    ):
044         # La formula de Paul Graham
045
046         g = 0.0
047         b = 0.0
048
049         if
    this.ok.palabras.has_key(palab
    ra):
050             g = 2.0 *
    this.ok.palabras[palabra]
051
052         if
    this.spam.palabras.has_key(pal
    abra):
053             b =
    this.spam.palabras[palabra]
054
055         # En caso de que no haya
    palabras de un tipo...
056         resultado = 0.0
057
058         if (g + b) > 0:
059             resultado = max( 0.1,
060                 min(0.99,
061                     min(1.0, b /
    this.spam.num)
062                     /
063                     (min(1.0, g/this.ok.num) +
    min(1.0, b/this.spam.num))))
064
065         return resultado
066
067     def
    construye_diccionario(this):
068         # Creamos un conjunto con las
    llaves
069         llaves =
    sets.Set(this.ok.palabras.keys
    ()) |
    sets.Set(this.spam.palabras.ke
    ys())
070
071         for llave in llaves:
072             this.probs_spam[llave] =
    this.calcula_prob_spam(llave)
073
074     def
    carga_correo(this,correos,corr
    eo):
075         # Carga las palabras de un
    correo en
076         # el diccionario
    correspondiente
077
078         for palabra in
    correo.split():
079             if
    correos.palabras.has_key(palab
    ra):
080                 correos.palabras[palabra]
    += 1
081             else:
082                 correos.palabras[palabra]
    = 1
083
084             def
    prob_spam_palabra(this,palabra
    ):
085                 if
    this.probs_spam.has_key(palabr
    a):
086                     return
    this.probs_spam[palabra]
087                 else:
088                     return 0.01
089
090         def es_spam(this, texto):
091             lista = texto.split()
092             probs = map(lambda x:
    this.prob_spam_palabra(x),
    lista)
093
094             prod = reduce(lambda x,y:
    x*y, probs)
095
096             resultado = prod / (prod +
    reduce(lambda x,y: x*y,
    map(lambda x: 1 - x, probs)))
097
098             return resultado
099
100         # Ejemplo de uso
101         f = filtro_bayesiano()
102         cadenas = ["The VigraMax is
    the best solution",
103             "hola Paul, ya tengo el
    artículo de Linux Magazine"]
104         for i in cadenas:
105             print f.es_spam(i)

```