INTELIGENCIA ARTIFICIAL

Filtrado de correo SPAM mediante inferencia probabilística

17/06/2016

Miembros

Menéndez Montes, Eva

Ramos González, José Renato

Índice

Descripción del proyecto

El objetivo de este trabajo es implementar un algoritmo de filtrado de correo SPAM mediante un proceso de inferencia probabilística. Para alcanzar dicho objetivo, en cuanto a las herramientas empleadas para la implementación del algoritmo, usando Python 3.5, hemos decidido utilizar Jupyter Notebook 4.0.6, ya que es el entorno utilizado en las prácticas con el cual ya estamos más que familiarizados.

La entrega de este trabajo se compone de un pdf con la documentación, un archivo de extensión .ipynb que contiene la implementación, y que ha de abrirse con Jupyter Notebook, y cinco archivos con extensión .mbox que son usados por el notebook. De los cuales, 3 de ellos han sido proporcionados junto con el enunciado del problema, mientras que los otros 2 archivos, new_ham.mbox y new_spam.box, contienen los dos primeros correos y los dos últimos correos de news.mbox, respectivamente.

Por último, hemos desarrollado el proyecto en un repositorio público alojado en GitHub, https://github.com/pastahito/spam-filter, por lo que, si se desea, se puede descargar el repositorio y ver la evolución del proyecto navegando a través de los distintos commits que hemos ido haciendo.

Enfoque del problema

Debido a la libertad en cuanto al diseño e implementación del algoritmo, por motivos académicos, hemos intentado darle un enfoque funcional al trabajo, en lugar de orientarlo a clases. De manera que, en la medida de lo posible, hemos dividido el algoritmo en pequeños trozos que se encarguen de una tarea en concreto, sin modificar sus entradas, es decir, funciones puras, para luego unirlos en un procedimiento que realice una tarea más compleja y que, a su vez, formará parte del algoritmo general.

Pues bien, el trabajo consta de tres partes: entrenamiento, clasificación e incorporación. Cada parte la realiza una función a la que llamamos procedimiento, y cada procedimiento se divide en pequeñas

funciones de utilidad. De manera que, solo son tres procedimientos los que ponemos a disposición para su uso: training(), clasification() e incorporation().

El procedimiento training() se encargará de crear el conjunto de entrenamiento a partir de correos ham y spam, y devuelve dos diccionarios con las cuentas sobre este conjunto, para así poder calcular las probabilidades que hagan falta posteriormente.

El procedimiento clasification() usa los diccionarios devueltos en el entrenamiento y actua sobre una serie de correos clasificándolos individualmente como spam o ham.

El procedimiento incorporation() agrega nuevos correos al conjunto de entrenamiento de spam o ham, escribiendo los correos en el fichero spambox o hambox, y actualizando los diccionarios devueltos en el entrenamiento, de manera que no hace falta vovler a ejecutar tranining() para clasificar nuevos correos tras la incorporación.

Procedimientos

En adelante, nos referiremos a las estructuras de datos por sus nombres traducidos en español. De manera que usaremos diccionarios, conjuntos, tuplas y listas en lugar de dictionary, sets, tuples y lists. Otras palabras reservadas como tipo de datos y demás, no los traducimos.

I. Entrenamiento

Se divide en tres tareas y un procedimiento:

- Separar un texto en palabras diferentes.
- Devolver el texto del título y cuerpo del mensaje al iterar sobre un mbox.
- Devolver el número de correos de un mbox junto a un diccionario de palabras con el número de correos en los que aparece.
- Devolver dos diccionarios, uno para spambox y otro para hambox, y el número de correos que contiene cada mbox.

A. separate_in_words(text, delimiter):

Devuelve un conjunto de strings (colección desordenada sin duplicados).

Text es un string cualquiera, y delimiter es una expresión regular. Depende del método .split() del módulo re. Funciona de la siguiente manera:

1. Separa text usando delimiter como criterio de separación.
2. Devuelve un conjunto con los trozos separados.

B. words_per_mail(mbox, delimiter):

Devuelve generator que hace yeild de un conjunto de palabras por cada correo de mbox.

Mbox es un string con la ruta de un .mbox, y delimiter es una expresión regular. Funciona de la siguiente manera:

1. Recorre cada correo contenido en el mbox con el método .mbox() del módulo mailbox.
2. Concatena el título y el cuerpo de dicho correo.
3. Utiliza la concatenación y delimiter en separate_in_words().
4. Hace yield del conjunto que devuelve 3. Al retomar la ejecución de este generador se vuelve al paso 1, si aún hay correos por recorrer.

C. training_dict(mbox, delimiter):

Devuelve una tupla que contiene el número de correos del mbox y un diccionario con palabras de tipo string como keys, y como values, el número de correos del mbox en el que aparece la palabra.

Mbox es un string con la ruta de un .mbox, y delimiter es una expresión regular. Funciona de la siguiente manera:

1. Recorre el mbox con el generator words_per_mail().
2. En un diccionario inicializado vacío, para cada palabra del conjunto enviado por el yeild, se crea una entrada nueva con value 0 si el diccionario no contenía la palabra, o se incrementa en 1 su value.
3. Devuelve una tupla con el diccionario y el número de veces que se ha recorrido el mbox.

D. training(spambox, hambox, delimiter = '\W+'):

Devuelve una tupla que contiene dos diccionarios, uno para spambox y otro para hambox, y el número de correos que contiene cada mbox.

Spambox es un string con la ruta de un .mbox que contiene únicamente correos spam y hambox es un string con la ruta de otro .mbox que contiene únicamente correos ham. El parámetro opcional, delimiter, es una expresión regular inicializada por defecto a cualquier secuencia no alfanumérica, el valor que tome delimiter es usado por las tres funciones descritas arriba.

1. Hace una llamada a training_dict() con spambox como su parámetro mbox.
2. Hace otra llamada a training_dict() con hambox como su parámetro mbox.
3. Devolvemos una tupla de 4 elementos, los devueltos en los pasos 1 y 2.

II. Clasificación

Se divide en dos tareas y un procedimiento:

- Devolver una lista de máximo n palabras con mejor clasificación individual.
- Devolver la probabilidad P(y|x1, ..., xn) con suavizado aditivo de Laplace.
- Devolver una lista de 0 o 1 por cada correo nuevo si es ham o spam, respectivamente.

Estas funciones usan una versión despejada de las fórmulas provistas en el enunciado del trabajo. Se adjuntan en el anexo al final del documento.

A. best_N_words(words, spam_dict, ham_dict, n):

Recibe un conjunto de palabras de un correo, los diccionarios de spam y ham y el número máximo de palabras que debe devolver. Devuelve una lista de máximo n palabras con mejor clasificación individual.

1. Se recorre el conjunto words.
   - Se obtiene su número de ocurrencias spam y ham en el conjunto de entrenamiento.
   - Se calcula P(y|x) para esa palabra usando la fórmula del anexo A, siendo 0.5 si el número de ocurrencias tanto en ham como en spam es 0.
   - Se calcula el valor absoluto de la diferencia entre esa probabilidad y 0.5. Ahora, las mejores clasificaciones son las que están más cercanas a 0.5.
2. Por último, ordenamos el diccionario de probabilidades y palabras en orden decreciente de probabilidad, y devolvemos las n primeras palabras.

B. naive_bayes(bests, spam_dict, s, ham_dict, h, debug):

Recibe la lista de las 15 mejores palabras, el diccionario de spam y ham, el número total de correos SPAM y el de correos HAM, y una variable booleana debug, que de ser True imprimirá en pantalla información relativa a la clasificación. Devuelve el valor $P(y|x_1, \ldots, x_n)$ para las palabras de la lista bests.

1. Se recorren las palabras de la lista de las 15 mejores.
   - Se calcula el número de ocurrencias spam y ham de la palabra.
   - Si alguno de los valores anteriores es 0, tenemos que realizar el suavizado, creando una entrada ficticia por cada 0 que tengamos.
   - Calculamos el producto del número de ocurrencias tanto para ham como para spam, ya que los necesitamos según la fórmula del anexo B.
2. Actualizamos los valores de s y h teniendo en cuenta el número de veces que ha tenido que realizarse el suavizado, y devolvemos la probabilidad de esas 15 palabras, calculada con la fórmula del anexo B.

C. clasification(newsbox, spam_dict, s, ham_dict, h, regex = '\W+', n = 15, debug = False):

Recibe un mbox de correos nuevos, los diccionarios de ham y spam, el número total de correos SPAM y HAM, y una expresión regular inicializada a cualquier cadena no alfanumérica, un número n inicializado a 15 y una variable debug inicializada a False. Devuelve una lista de 0 o 1 por cada correo nuevo indicando si es HAM o SPAM, respectivamente.

1. Se recorren los conjuntos de palabras de los correos del mbox.
   - Se obtienen las 15 palabras con mejor clasificación individual, usando best_N_words(words, spam_dict, ham_dict, n).
   - Se calcula el valor $P(y|x_1, \ldots, x_n)$ para las 15 palabras obtenidas anteriormente, usando naive_bayes(bests, spam_dict, s, ham_dict, h, debug).
   - Se clasifica el correo como spam añadiendo un 1 a una lista si $P(y|x_1, \ldots, x_n)$ es superior a 0'9, en caso contrario se añade 0 para que sea ham.
2. Se devuelve la lista de 1s y 0s.

Si la variable debug es True, se imprimirá en pantalla información relativa a los cálculos realizados en naive_bayes().

III. Incorporación

Se divide en dos tareas y un procedimiento:

- Escribir los correos nuevos en un mbox del conjunto de entrenamiento.
- Devolver el nuevo número de correos del conjunto de entrenamiento y un nuevo diccionario actualizado.
- Agrupar las 2 tareas anteriores en un procedimiento.

A. append_new_to_mbox(new_mbox, mbox):

Recibe por parámetro dos rutas de mbox, 'new_mbox', donde se encuentran los correos a añadir, y 'mbox', donde van a ser añadidos.

El procedimiento que se sigue es el siguiente:

1. Habilitar la escritura de mbox en modo "appending". Todo lo que se escriba en este documento se añade al final.
2. Habilitar la lectura de new_mbox.
3. Se escribe en mbox el contenido de new_mbox, y se cierran ambos archivos.

B. update_dict_and_size(new_mbox, dictionary, size, delimiter):

Con esta función se actualiza size y dictionary, teniendo en cuenta las palabras de los correos pertenecientes a new_mbox.

1. Hacemos una copia de size y dictionary que devolveremos.
2. Por cada conjunto de palabras al iterar new_mbox con words_per_mail(new_mbox, delimiter):
   - En la copia de dictionary, para cada palabra del conjunto enviado por el yeild, se crea una entrada nueva en la copia con value 0 si no contenía la palabra, o se incrementa en 1 su value en caso contrario.
   - Incrementamos la copia de size en 1.
3. Devolvemos las copias actualizadas de size y dictionary.

C. incorporation(new_mbox, mbox, size, dictionary, delimiter = '\W+'):

New_mbox es un string con la ruta de un .mbox que contiene únicamente correos nuevos a incorporar al conjunto de entrenamiento. mbox es un string con la ruta de otro .mbox en el que se

guardarán los correos nuevos. Size es el número de correos actual que hay en mbox, y dictionary su diccionario asociado. El parámetro opcional, delimiter, es una expresión regular inicializada por defecto a cualquier secuencia no alfanumérica, el valor que tome delimiter es usado por update_dict_and_size().

1. Ejecuta append_new_to_mbox(new_mbox, mbox).
2. Devuelve update_dict_and_size(new_mbox, dictionary, size, delimiter).

Aportaciones al trabajo

1. Enfoque funcional del proyecto, al dividir el algoritmo en pequeños trozos que se encargan de una tarea en concreto, sin modificar sus entradas, es decir, funciones puras, para luego unirlos en un procedimiento que realiza una tarea más compleja. No usamos clases.
2. Las 5 rutas a los ficheros .mbox son configurables, para que el uso de estos no esté sujeto a tener los ficheros en la misma ruta relativa que nosotros.
3. La expresión regular delimiter es configurable, usada para separar un texto en trozos que llamamos palabras. Basta con pasar por parámetro delimiter = nuevo_regex a los procedimientos training() e incorporation(), para que los separe de otra manera,
4. El número n es configurable, usado como máximo número de palabras de mejor clasificación individual para hacer Naive Bayes. Basta con pasar por parámetro n = nuevo_maximo al procedimiento clasification(), para modificar su valor.
5. Hemos calculado Naive Bayes usando un suavizado aditivo de Laplace.
6. Incluimos en clasification() un parámetro opcional llamado debug inicializado a False, que cuando es true imprime en pantalla información sobre S, H, Swi, Hwi, ∏Sw, ∏Hw y $P(y|x_1, \ldots, x_n)$ por cada correo nuevo a clasificar.
7. En el enunciado del trabajo se pedía distinguir un procedimiento para incorporar 1 spam y otro para 1 ham. El procedimiento incorporation() los distingue dependiendo el parámetro mbox que se le pase e independientemente de la cantidad de correos nuevos que contengan, y actualiza las variables del número de correos ham y spam, además de actualizar los diccionarios. De manera que, tras el uso de incorporation() no hace falta volver a ejecutar training() para poder clasificar un nuevo conjunto de correos nuevos con clasification().

Anexo

A. Bayes:

$$P(y|x) = \frac{P(y)P(x|y)}{P(y)P(x|y) + P(\neg y)P(x|-y)} = \frac{\frac{S}{S+H} \times \frac{S_w}{S}}{\frac{S}{S+H} \times \frac{S_w}{S} + \frac{H}{S+H} \times \frac{H_w}{H}} = \frac{S_w}{S_W + H_W}$$

B. Naive Bayes:

$$P(y|x_1,\dots,x_n) = \frac{P(y)\prod_i P(x_i|y)}{P(y)\prod_i P(x_i|y) + P(\neg y)\prod_i P(x_i|\neg y)}$$

$$= \frac{\frac{S}{S+H}\left[\frac{S_{w_1}}{S}\times\frac{S_{w_2}}{S}\times\dots\times\frac{S_{w_n}}{S}\right]}{\frac{S}{S+H}\left[\frac{S_{w_1}}{S}\times\frac{S_{w_2}}{S}\times\dots\times\frac{S_{w_n}}{S}\right] + \frac{H}{S+H}\left[\frac{H_{w_1}}{H}\times\frac{H_{w_2}}{H}\times\dots\times\frac{H_{w_n}}{H}\right]}$$

$$= \frac{S\left[\frac{\prod_i S_{w_i}}{S^n}\right]}{S\left[\frac{\prod_i S_{w_i}}{S^n}\right] + H\left[\frac{\prod_i H_{w_i}}{H^n}\right]} = \frac{\prod_i S_{w_i}}{S^{n-1}\left[\frac{\prod_i S_{w_i}}{S^{n-1}} + \frac{\prod_i H_{w_i}}{H^{n-1}}\right]}$$

$$= \frac{\prod_i S_{w_i}}{\prod_i S_{w_i} + \prod_i H_{w_i}\times\left(\frac{S}{H}\right)^{n-1}}$$

Bibliografía

1. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla. Guía de Referencia Rápida de Python. <http://www.cs.us.es/cursos/iais-2015/practicas/Guía de referencia de Python.pdf>.
2. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla. Introducción a Python. <http://www.cs.us.es/cursos/iais-2015/practicas/introPython.pdf>.
3. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla. Aprendizaje Automático. <http://www.cs.us.es/cursos/iais-2015/temas/tema-06.pdf>.
4. Wikipedia. Additive Smoothing. <https://en.wikipedia.org/wiki/Additive_smoothing>.
5. Python 3.5.2rc1 documentation. re — Regular expresion operations. <https://docs.python.org/3.5/library/re.html#re.split>.
6. Python 3.5.2rc1 documentation. mailbox — Manipulate mailboxes in various formats. <https://docs.python.org/3.5/library/mailbox.html#mbox>.
7. Python 3.5.2rc1 documentation. Generators — Functional Programming HOWTO. <https://docs.python.org/3/howto/functional.html#generators>.