Detección automática del idioma de un texto mediante el uso de unigramas y 2-gramas

Automatic Language Detection by Means of Unigrams and 2-grams

Lic. Manuel Vázquez Acosta*[manuelva@uci.cu] Roberto O Labrada Sedeño [rolabrada@estudiantes.uci.cu]
Miguel Y Godales Cruz [mygodales@estudiantes.uci.cu]
Universidad de las Ciencias Informáticas

Enero de 2008

Resumen

En este documento se describe un algoritmo utilizado para la determinación automática del idioma de un texto. Se explica la implementación realizada de este algoritmo y se analiza su complejidad temporal. Por último, se debate la aplicabilidad de esta implementación en proyectos reales.

In this paper we describe an algorithm for automatic detection of languages in text documents. We also discuss our current implementation of this algorithm and assess its temporal complexity. Finally, we debate the applicability of this implementation in real projects.

Palabras claves: Clasificación, Experimentación, Implementación, Procesamiento de Lenguaje Natural, Python.
Keywords: Assorting, Experimentation, Implementation, Natural Language Processing, Python.

1. Introducción

La clasificación/agrupación de texto es un área de investigación activa y en la Universidad de las Ciencias Informáticas está alcanzando cierta relevancia (Camps Díaz and Vázquez Acosta, 2007). Muchos de estos algoritmos usan un modelo vectorial para representar los documentos basado en el que se usa en los algoritmos para la recuperación de información (Manning et al., 2007, cf. Cap. 6)

Un área en concreto dentro de la clasificación de textos es la determinación del idioma. Algunas aplicaciones públicas como los servicios que brinda Google hacen uso evidente de este tipo de clasificación.

En este artículo describimos un algoritmo relativamente simple para la determinación automática del idioma de un texto. En la sección 2.1 presentamos un modelo vectorial para la representación de los documentos y describimos el algoritmo. Más adelante en la sección 2.2 nos concentramos en la implementación de ese modelo. En la sección 3.2 exponemos los resultados experimentales sobre una colección de textos. Finalmente, en la sección 3.3 debatimos la aplicación de esta implementación en proyectos reales.

2. Materiales y métodos

En esta sección introducimos los conceptos fundamentales que forma la base del algoritmo que se presenta.

*Enviar correspondencia a Lic. Manuel Vázquez Acosta

2.1. Modelo vectorial

El problema de determinar el idioma de un texto A puede ser resuelto si logramos determinar si A y otro documento B están en el mismo idioma y conocemos el idioma de B.

Esta formulación del problema implica que debemos tener una manera para comparar los documentos A y B en relación a su idioma.

En la disciplina de Recuperación de Información se usa una representación vectorial (Manning et al., 2007) de cada documento para determinar la similitud entre un documento y la consulta realizada por el usuario.

Cada documento es almacenado como un vector en un espacio n-dimensional, y cada componente de ese vector es calculado a partir del documento original.

Una variante bastante establecida es tener un componente del vector por cada *término* que haya en la *colección* de documentos. De esa forma el componente x_t está relacionado con el término t, y generalmente x_t es directamente proporcional a la frecuencia del término t en el documento.

La similitud entre dos documentos se define por la fórmula del coseno del ángulo entre los vectores que representan esos documentos:

$$\cos\theta = \frac{d_1 \cdot d_2}{\mid d_1 \mid\mid d_2 \mid} \tag{1}$$

En esta fórmula d_1 y d_2 son los vectores de los documentos a comparar, el operador $\cdot$ es el producto escalar de dos vectores y $\mid d \mid$ es el módulo del vector d.

También se puede usar la distancia entre los vectores normalizados:

$$\operatorname{diff}(d_1, d_2) = \left| \frac{d_1}{\mid d_1 \mid} - \frac{d_2}{\mid d_2 \mid} \right| \tag{2}$$

En el primer caso tenemos una fórmula para calcular la *similitud* y en el segundo una fórmula para la *diferencia*. Mientras más similares sean dos documentos menos diferentes serán. En particular, se puede observar que se cumple que $\operatorname{diff}(d_1, d_2) = 0$ si y sólo si $\cos\theta = 1$.

La figura 1 muestra la relación que existe entre estas dos medidas en un espacio de 2 dimensiones.

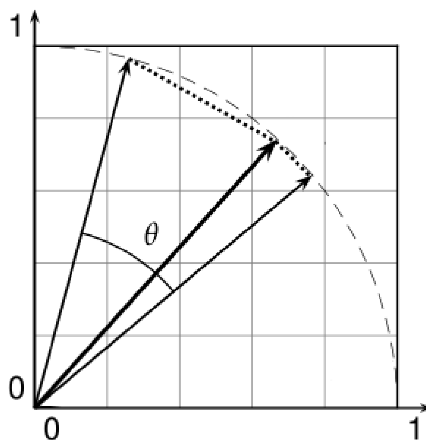

Figura 1: Relación entre el “ángulo” y la distancia entre vectores normales en un espacio de dos dimensiones

2.1.1. Modelo vectorial para la detección del idioma

Para poder aplicar el modelo vectorial necesitamos un esquema para determinar los vectores asociados a un documento. Los esquemas aplicados en la recuperación de información no son directamente aplicables a la determinación del idioma porque se basan en los términos; y en el mismo idioma existen documentos escritos con términos diferentes.

La probabilidad de ocurrencia de los n-gramas[1] ha sido explotada con éxito en este sentido (Bastrup and Pöpper, 2003; Graham and Lapalme, 2005).

En este artículo usaremos la frecuencia de los unigramas y 2-gramas para describir cada documento, y también la ecuación 1 para comparar los documentos.

Por ejemplo, para el documento "Hola lola" con el alfabeto español, el vector tendría 33 posibles unigramas[2] y $33 \times 33 = 1089$ posibles 2-gramas. Por tanto, los vectores de cada documento tendrían 1122 componentes. Sin embargo, la mayoría de ellos tiene frecuencia cero en nuestro documento y por tanto sólo representaremos los que son diferentes de cero:

Cuadro 1: Frecuencias de los unigramas y 2-gramas para el texto "Hola lola"

h	o	l	a	ho	ol	la	lo
$1/8$	$2/8$	$3/8$	$2/8$	$1/6$	$2/6$	$2/6$	$1/6$

Note que las frecuencias de los unigramas se calculan en base a la cantidad total de unigramas del documento, y las frecuencias de los 2-gramas respecto a la cantidad total de 2-gramas del documento.

Una vez que tenemos el vector del documento al cual queremos determinar su idioma, se puede comparar con el vector que *representa cada idioma*.

El algoritmo decide que el documento está en el idioma representado por el vector que más cercano esté al vector del documento. Si $v_1, v_2, \ldots, v_n$ son los vectores que representan los idiomas $1, 2, \ldots, n$, entonces el algoritmo decidirá que d está en el idioma i si $\mathrm{diff}(v_i, d) = \min\{\forall j \ \mathrm{diff}(v_j, d)\}$. Alternativamente, se puede usar la ecuación (1), pero entonces se decidirá por la máxima simulitud.

2.2. Implementación

La implementación actual de este algoritmo está hecha en Python. Esta implementación solamente puede decidir si un texto está en inglés o español. Sin embargo, puede extenderse fácilmente para incorporar más lenguajes que usen el alfabeto latino. Los autores no han considerado la inclusión de otros alfabetos como el cirílico, árabe, sánscrito, hebreo, etc. Como se evidencia en (Graham and Lapalme, 2005) la inclusión de tales alfabetos no es trivial. Incluso con el alfabeto latino, nuestra implementación se ve afectada por la codificación (*charset*) de los textos como se demuestra en la sección 3.2.

2.3. Descripción del paquete

La implementación está en un paquete de Python llamado `langdet` que puede ser instalado en el sistema mediante el mecanismo estándar de distribución para Python: `distutils`.

[1] Un n-grama es una secuencia de n letras del alfabeto del lenguaje

[2] abcdefghjijlkmnñopqrstuvxyzáéíóúü

Dentro de este paquete tenemos los módulos `langdet.py` y `streams.py` junto al subpaquete `languages` que contiene los módulos con los descriptores (vectores) de cada lenguaje.

2.3.1. Interfaz del paquete

Luego de instalado el paquete su uso es muy simple. El código en la figura 2 muestra cómo usarlo para un texto fijo.

```
from langdet.langdet import LanguageDetector
from langdet.languages import spanish, english
from langdet.streams import Stream
from StringIO import StringIO

text = Stream(StringIO(u"Hola mundo. El día está precioso"))

detector = LanguageDetector()
if detector.detectLanguage(text) == spanish:
    print "El texto está en español"
else:
    print "El texto está en inglés"
```

Figura 2: Esquema de uso del paquete `langdet`

Por otra parte, el mismo módulo `langdet.py` puede usarse directamente desde otros programas, como se muestra en la figura 3.

```
$ echo "Hola a todo el mundo. El día está precioso" > text.txt
$ python langdet.py text.txt
es

$ echo "Hello world. The day is beautiful" > text.txt
$ python langdet.py text.txt
en
```

Figura 3: Uso de langdet como ejecutable

El único parámetro en este caso es el fichero que contiene el texto a procesar, y la salida es el identificador ISO 639-1 del lenguaje detectado.

Esto hace factible que otros programas que no tengan forma cargar módulos de Python puedan beneficiarse también de este paquete.

3. Evaluación y discusión

3.1. Análisis de la complejidad

Como la cantidad de unigramas y 2-gramas está acotada superiormente se usó un diccionario de Python para representar los vectores. Los diccionarios de Python son una estructura de datos que permite almacenar pares (llave, valor) con un tratamiento similar al de las listas (arreglos).

Esto facilitó la implementación del cómputo de la similitud del coseno y también el proceso de construcción de los módulos que contienen los vectores representativos de cada lenguaje. Si observa la implementación de esos módulos verá que simplemente declaran dos constantes:

`iso` Cuyo valor es el código ISO 639-1 del idioma

`freq` Cuyo valor es el diccionario que se usa para representar el vector del idioma.

Para detectar el idioma de un texto, primero se calcula su vector a partir de texto original y luego se compara con los vectores representativos del español e inglés.

El método `__computeFrequencies__` es el responsable de calcular el vector de frecuencias. El resultado de esta función es el diccionario que contiene los pares (n-grama, frec).

El vector de frecuencias no es normalizado porque sabemos que $\sum_i x_i = 2$ para todos los documentos, y por tanto $\mid d \mid = \sqrt{2}$, por lo que podemos simplificar la ecuación 1:

$$\frac{d_1 \cdot d_2}{\mid d_1 \mid\mid d_2 \mid} = \frac{d_1 \cdot d_2}{2} \sim d_1 \cdot d_2 \tag{3}$$

Al inspeccionar el método `__cosineSimilarity__` podrá ver que es esta última expresión la que se calcula (no se efectúa la división).

Se puede observar que este método tienen una complejidad temporal que depende de la cantidad de unigramas y 2-gramas del alfabeto y no de la longitud del documento, y por tanto, su complejidad es del orden $O(1)$:

Dado el alfabeto Σ, si $\Gamma \subseteq \Sigma \cup \Sigma^2$ es el subconjunto de unigramas y 2-gramas del alfabeto que se usan en los textos, el orden de complejidad de esta función es $O(\mid \Gamma \mid)$. Como $\mid \Gamma \mid$ está limitado superiormente por $\mid \Sigma \cup \Sigma^2 \mid$ y este conjunto es finito, entonces podemos afirmar que esta función corre en tiempo constante con relación al documento.

El método `__computeFrequencies__` es el que determina la complejidad de esta implementación. El orden de este método es $O(n)$ donde n es la cantidad de caracteres del texto.

3.2. Evaluación del algoritmo

Para extraer los vectores representativos de cada idioma se usaron solamente dos textos que los autores consideramos que eran suficientes para describir cada lenguaje.

Para el español se usó el tomo primero de la novela "*El ingenioso hidalgo Don Quijote de La Mancha*" del escritor Miguel de Cervantes. Para el idioma inglés se usó la edición King James de "*The Holly Bible*".

En esta sección describimos los resultados de correr el algoritmo entrenado con estos textos en un corpus de documentos extraídos de (Koehn, 2005). Esta colección tiene 656 documentos en cada idioma, que suman un total de 418 mega-bytes en texto plano. Los documentos en español contienen un total de 37 870 751 palabras, mientras que los documentos en inglés contienen 36 429 274 palabras.

Para correr el algoritmo se usó la variante de ejecutar el módulo directamente desde la consola de un sistema GNU/Linux. La flexibilidad del `bash` y del comando `find` ayudaron a correr el algoritmo para todo el corpus:

```
$ find es-en/es/ -type f -exec python langdet.py {} \; |grep es |wc -l
```

Con esta sencilla orden se ejecutó el programa `langdet.py` para cada documento en español de la colección y se contaron las respuestas "es". De forma similar se procedió para contar los aciertos para el idioma inglés.

Para los textos en español de la colección se obtuvo un 100 % de aciertos. Para los textos en inglés falló sólo para 2 documentos (99, 70 %). Sin embargo, el fallo no fue en la detección del idioma en sí, sino en la detección de la codificación (*charset*) del fichero. Siempre que se pudo detectar la codificación del fichero con un mínimo del 80 % de confiabilidad, se detectó el idioma satisfactoriamente. En (Graham and Lapalme, 2005) se abunda sobre la relación entre lenguaje y codificación y se opta por una solución de determinación conjunta. El algoritmo presentado en este artículo, sin embargo, se abstrae de la determinación de la codificación, la cual es realizada por el paquete `chardet` de Python.

El tiempo de ejecución fue de 1 647 segundos (27 minutos) para procesar los 418 mega-bytes de texto del corpus. Lo que da una razón de 15,48 Mb/min.

Los autores pensamos que el modelo se puede extender de modo que podamos determinar, con cierta certeza, el momento en que no es necesario continuar leyendo del fichero y proceder con un vector construido con una porción del documento. En un experimento inicial los autores probaron que usando sólo los primeros 4 Kb del fichero se redujo al 10 % el tiempo de ejecución. No obstante, aumentaron los errores en la detección de la codificación tanto en inglés como en español. Aunque el beneficio obtenido es mucho mayor que los problemas introducidos, los autores pensamos que se puede obtener un procedimiento que reduzca el tiempo de ejecución y no introduzca estos errores.

3.3. Aplicabilidad

Las aplicaciones de este paquete en proyectos reales son varias y ya hemos apuntado a algunas de ellas en la introducción de este artículo. En esta sección debatimos la posibilidad real de integrar este producto con otros en desarrollo.

Debido a que el Python es un lenguaje portable (y portado) a la mayoría de las plataformas para las que se produce en la Universidad de las Ciencias Informáticas, es fácil incluir este producto en otros proyectos. Aunque el Python tiene una participación marginal en la producción, este módulo se puede usar desde programas escritos en otros lenguajes como se explicó previamente. Basta con ejecutar el fichero `langdet.py` y obtener la respuesta.

Esta solución, sin embargo, no es muy elegante en algunos casos porque se necesita generar un fichero texto con el contenido a clasificar. Estos ficheros pudieran quedar en el disco duro de la máquina en lugares no protegidos y ser fuentes de escapes de información.

En GNU/Linux esto puede resolverse usando un sistema de ficheros temporal (ramfs o tempfs). En Windows, los autores no conocen una solución directa, por lo que se pretende extender el módulo `langdet.py` para que acepte el texto por la entrada estándar, de modo que exista una comunicación directa entre los dos procesos.

4. Conclusiones y trabajo futuro

La detección del idioma de un lenguaje entre los idiomas inglés y español se puede realizar con un alto grado de confianza mediante la representación vectorial del documento basada en la frecuencia de los unigramas y 2-gramas en el documento y la comparación mediante la fórmula de la similitud del coseno.

El tiempo de ejecución del algoritmo varía linealmente con relación al tamaño del documento, y para volúmenes de texto moderado es aceptable. Para grandes volúmenes de texto, debemos estudiar un modo de reducir ese tiempo. De forma alternativa, los clientes del módulo pudieran entregar sólo una porción del documento, que respete la codificación original del documento.

La implementación realizada es portable y aplicable de forma inmediata en los proyectos que requieran de este tipo de clasificación de textos.

La seguridad para usar el programa desde otro que no entienda el Python puede verse comprometida en ciertas circunstancias, especialmente en la plataforma Windows.

Referencias

Sofia Bastrup and Christina Pöpper. Language detection based on unigram analysis and decision trees. Technical report, Lund University, 2003.

Darel Camps Díaz and Manuel Vázquez Acosta. Implementación de algoritmos de agrupamiento. Technical report, Universidad de las Ciencias Informáticas, 2007. URL `http://bibliodoc.uci.cu/TD/TD_0276_07.pdf`.

Russell Graham and Guy Lapalme. Automatic identification of language and encoding. Technical report, Recherche appliquée en linguistique informatique, Université de Montréal, 2005.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Submit*, 2005. URL `http://www.statmt.org/europarl/`.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. 2007. URL `http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf`.