Detección de reutilización de código fuente entre lenguajes de programación en base a la frecuencia de términos

Enrique Flores, Alberto Barrón-Cedeño, Paolo Rosso, and Lidia Moreno

Universidad Politécnica de Valencia, Dpto. de Sistemas Informáticos y Computación
Camino de Vera s/n, E-46022 Valencia, España
{eflores,lbarron,prosso,lmoreno}@dsic.upv.es
http://www.dsic.upv.es/

Resumen En la actualidad, hay muchos repositorios públicos donde cualquiera puede acudir y obtener un código fuente o parte de él y utilizarlo en sus programas sin permiso del autor o sin citarlo. Otro caso más común se da en el mundo académico, ya que cuando varios estudiantes deben de realizar el mismo trabajo algunos de ellos pueden copiar o modificar el trabajo de los compañeros para que parezca un trabajo diferente.
En este artículo proponemos un sistema que intenta detectar la reutilización de código fuente incluso entre lenguajes de programación distintos, estudiando la influencia de los comentarios, los nombres de variables y las palabras del lenguaje. Las técnicas aplicadas para la detección de la similitud han sido Frecuencia de Términos (TF) y Frecuencia Inversa (TF-IDF) considerando como términos los n-gramas de caracteres.

Keywords: Reutilización de Código Fuente, Detección de Plagio Multilíngüe, Procesamiento del Lenguaje Natural

1. Introducción

La popularización del uso de Internet ha posibilitado la existencia de enormes cantidades de información al alcance de cualquier usuario. Las obras publicadas en este medio digital están expuestas a copias y reutilización de todo o parte de ellas. Por ello, existe un gran interés en identificar la autoría y el posible plagio en las obras publicadas. Sin embargo, no es trivial la localización y comparación de toda la información relevante para asegurar un plagio, o al menos cierto grado de similitud entre obras.

La ingeniería lingüíística, más conocida en el área de la Inteligencia Artificial como Procesamiento del Lenguaje Natural (PLN), facilita el tratamiento automático de la documentación textual. Un desafío interesante cosiste en aplicar recursos y técnicas de PLN con el fin de detectar similitud, incluso para texto traducido entre distintas lenguas [8].

Actualmente, la disponibilidad de código fuente en la red es muy amplia facilitando la reutilización total o parcial de programas que han sido previamente

implementados y testeados por el propio autor o bien por autores externos. A esta práctica común se le conoce como reuso"de código fuente (source code reuse). Considerando el código fuente como obras con autoría, su uso por programadores ajenos, al menos debería realizarse citando siempre las fuentes o autoría de las mismas. Un campo de especial aplicación de los problemas expuestos se da en el ámbito académico; medio propenso a la copia de las tareas correspondientes a una determinada materia por parte del alumnado.

Otra forma de reuso de código fuente consiste en encontrar dentro de un repositorio de código fuente algún algoritmo implementado en un lenguaje de programación y traducirlo a otro lenguaje de interés para el programador que ha realizado la búsqueda. La disponibilidad de código fuente traducido a otro lenguaje de programación puede dar lugar a una nueva línea de investigación dentro del campo de detección de autoría o detección de plagio. Para dar solución a este tipo de problema, se debería disponer de sistemas con capacidad de detección de un código fuente similar a otro escrito en un lenguaje de programación diferente.

La carencia de estudios dirigidos a este tipo de reuso de código fuente traducido entre diferentes lenguajes de programación ha motivado la realización de este trabajo. Los autores pretenden demostrar que es posible detectar la similitud de código fuente monolíngüe o multilíngüe aplicando técnicas de PLN a través de los experimentos realizados.

2. Método propuesto

El método que vamos a utilizar para la detección de similitud entre códigos fuente es *term frequency* (tf). Este método consiste en dividir el documento que contiene el código fuente en términos, es decir en n-gramas de carácteres, calcular la frecuencia de aparición de esos términos y normalizar dichas frecuencias de forma que todas ellas sumen 1 con el fin de poder comparar documentos de diferentes tamaños. Finalmente, para poder comparar dos documentos se ha aplicado la similitud del coseno .

Para poder probar esta propuesta se ha hecho uso de una colección de documentos que contienen programas en C++, Java y Python de un Sistema Multiagente (MAS), que permiten desarrollar agentes con sus respectivos comportamientos. Para cada lenguaje tenemos una colección de programas que tienen cierta correspondencia con programas en los otros lenguajes. Esta correspondencia puede ser total o parcial en funcionalidad.

3. Experimentación

El objetivo general de este trabajo es proponer un método para detectar similitud entre códigos fuente escritos en distintos lenguajes de programación, es decir multilingüe. Para ello se han realizado los experimentos sobre los lenguajes C, Java y Python con el corpus SPADE de la siguiente forma. Se ha utilizado distintos tipos de comparación como son: el texto entero, el texto sin los comentarios, solamente los comentarios, el texto sin las palabras del lenguaje, las

palabras del lenguaje únicamente y el texto sin los comentarios y sin las palabras del lenguaje.

Estos experimentos se han realizado en base a term frequency (tf) y para term frequency-inverse document frequency (tf-idf) para n-gramas de caracteres con valores de N de 1 a 5.

En la mayoría de los experimentos entre los pares de lenguajes de programación mencionados se han obtenido los mejores resultados teniendo en cuenta el texto entero y el texto entero sin los comentarios con los mismos valores en ambos casos. Además los mejores valores de n-gramas de carácteres han sido de tamaño 3 hasta 5. En términos de promedio y desviación de la posición de los documentos correspondientes a los códigos fuente utilizados los mejores resultados para trigrammas han sido de 1.00 ± 0.00 entre Java y C++, de 1.44 ± 0.83 entre Python y C++, y de 1.62 ± 1.10 entre Java y Python.

Otro resultado interesante que hemos detectado ha sido el que tf-idf funcionara de manera bastante pareja a tf, dado que para aplicar tf-idf se necesita conocer el corpus de referencia y que hay que realizar un preproceso de éste para obtener la tabla de frecuencias de términos. Esto puede ser debido al tamaño del corpus, por lo que habría que comprobarlo con un corpus mucho más grande del utilizado en este estudio previo.

4. Conclusiones

Después de realizar todos los experimentos, una primera conclusión que se puede extraer es que tf-idf funciona bastante similar a tf. En el futuro se espera volver a realizar los experimentos con un corpus más grande con el fin de comprobar si se producen cambios al respecto. De no ser así, la opción más conveniente será utilizar tf dado que no se necesita ningún corpus de referencia y ahorra tiempo de cálculo.

Los comentarios no parecen relevantes a la hora de determinar la similitud, ya que en la mayoría de casos funciona mejor el texto entero y el texto entero sin los comentarios para valores de 3 o más n-gramas de caracteres. Esto sugiere que por lo que se pueden obviar, ganando tiempo de cálculo y evitando manipulaciones por parte del usuario en los comentarios en el nivel 1 sugerido por Faidhi en [4]. Además, que los mejores resultados se consiguan con 3 o más n-gramas de carácteres sugiere que se puede detectar el estilo de programación al igual que sucede en escritura como se comenta en [10].

El hecho de que los resultados sean similares con trigramas que con tetragramas y pentagramas, nos hace pensar que el sistema se ha estabilizado y que no va a mejorar más. Si partimos de este supuesto, convendría trabajar con el tamaño mínimo de n-grama dado que es menos costoso el cálculo de éstos n-gramas.

El presente trabajo se considera como unas pruebas preliminares con la intención de comprobar los métodos y técnicas de similitud de textos que pueden aplicarse a la similitud de código fuente. Se considera que aún quedan muchas más por evaluar como las ventanas deslizantes [10]: (explicar en nota al pie de

página) o como las técnicas CL-Explicit Semantic Analysis[8] y CL-Alignment-based Similarity Analysis[7], utilizadas entre distintos idiomas del lenguaje natural. Por otra parte, se considera que este experimento se ha realizado con un corpus muy pequeño, tanto en cantidad de documentos a comparar como en lenguajes de programación estudiados. En un corto plazo de tiempo se espera poder disponer de un corpus más amplio y que cuente con una mayor variedad de lenguajes de programación.

Agradecimientos Este trabajo ha sido desarrollado con la ayuda del proyecto de investigación MICINN TEXT-ENTERPRISE 2.0 TIN2009-13391-C04-03 (Plan I+D+i)

Referencias

1. Arwin, C. and Tahaghoghi S.M.M. : Plagiarism Detection across Programming Languages. Proceedings of the 29th Australasian Computer Science Conference vol. 48, pp. 277–286). Darlinghurst, Australia: Australian Computer Society. (2006)
2. Burrows, S., Tahaghoghi, S.M.M., and Zobel, J. (2006). Effi- cient plagiarism detection for large code repositories. Software?Practice and Experience, (37):151-175.
3. Clough, P. (2000). Plagiarism in natural and programming languages: An overview of current tools and technologies. Research Memoranda: CS-00-05. Department of Computer Science. University of Sheffield, UK.
4. Faidhi, J. and Robinson, S. (1987). An empirical approach for detecting program similarity and plagiarism within a university programming environment. Comput. Educ., 11(1).
5. Halstead, M. H. (1972). Natural laws controlling algortithm structure? SIGPLAN Noticies, 7(2).
6. Jankowitz, H. T. (1988). Detecting plagiarism in student pascal programs. The computer journal, 31(1).
7. Pinto, D., Civera, J., Barrón-Cedeño, A., Juan, A., and Rosso, P. (2009). A statistical approach to crosslingual natural language tasks. Journal of Algorithms, 64(1):51-60.
8. Potthast M., Barrón-Cedeño A., Stein B., Rosso P. Cross-Language Plagiarism Detection. In: Languages Resources and Evaluation. Special Issue on Plagiarism and Authorship Analysis, vol. 45, num. 1. DOI: 10.1007/s10579-009-9114-z
9. Prechelt, L., Malpohl, G., and Philippsen, M. (2002). Finding plagiarisms among a set of programs with jplag. Journal of Universal Computer Science, 8(11):1016-1038.
10. Stamatatos, E. (2009). Intrinsic Plagiarism Detection Using Character n-gram Profiles. In Stein et al. (2009), pages 38-46. URL `http://ceur-ws.org/` Vol-502.
11. Whale, G. (1990). Software metrics and plagiarism detection. Journal of Systems and Software, (13):131-138.
12. Wise, M.J., Detection of similarities in student programs: YAPing may be preferable to Plagueing, SIGSCI Technical Symposium, Kansas City, USA, pp(268-271), March 5-6, 1992.