AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1069.2022
VI Jornadas de Intercambio y Difusión de los Resultados de Investigaciones de los Doctorandos en Ingeniería

Detección automática de similitudes de código fuente utilizando técnicas de aprendizaje automático

Automatic detection of source code similarities using machine learning techniques

Presentación: 4-5/10/2022

Doctorando:

Marina Elizabeth Cardenas

Grupo de Investigación, Desarrollo y Transferencia en Aprendizaje Automático, Lenguajes y Autómatas- Centro de Investigación y Desarrollo de Software- Facultad Regional Córdoba - Universidad Tecnológica Nacional – Argentina
ing.marinacardenas@gmail.com

Director:

Julio Javier Castillo

Resumen

En el presente trabajo se plantea el desarrollo de un modelo para detección de similitudes de código fuente para poder determinar la existencia de prácticas de reutilización aplicando técnicas vinculadas al aprendizaje automático con un enfoque a la lingüística computacional. Existen diversas técnicas desarrolladas por diversos autores que permiten la detección de fragmentos de código fuente similares (usualmente llamados Clones de Código o Code Clones) enfocados en los distintos tipos de clones. La identificación de estos clones de código fuente puede servir para varios propósitos, entre los que se puede mencionar el estudio de la evolución del código fuente de un proyecto, detección de prácticas de reutilización, extracción de un fragmento de código para "refactorización" del mismo, detección y seguimiento de defectos, fallas y/o virus para su corrección, entre otros.

Palabras claves: código fuente, similitudes, reutilización, aprendizaje automático, texto, análisis.

Abstract

This paper proposes the development of a model to detect similarities in source code in order to determine the existence of reuse practices using techniques linked to machine learning with a focus on computational linguistics. There are various techniques developed by various authors that allow the detection of similar source code fragments (usually called Code Clones) focused on the different types of clones. The identification of these source code clones can serve several purposes, such as the study of the evolution of the source code of a project, detection of reuse practices, extraction of a code fragment for its refactoring, detection and monitoring of defects, failures and/or viruses for their correction, among others.

Keywords: source code, similarities, reuse, machine learning, text, analysis.

Introducción

Actualmente, la reutilización de código fuente es una práctica comúnmente utilizada en el proceso de desarrollo de software que conlleva una serie de ventajas, como así también, de inconvenientes que van desde la introducción de errores o defectos en el código fuente, lo que incurre en el incremento de tiempo de desarrollo y costos de mantenimiento (Fowler et al., 1999), lo cual afecta también a su calidad (Roy et al., 2009), hasta problemas éticos y legales por infringir derechos de propiedad intelectual.

Entre algunas de las principales motivaciones por las cuales se realiza la reutilización se pueden mencionar las siguientes (Rattan et al., 2013):

- Limitaciones en las capacidades técnicas de los programadores.
- Restricciones de tiempo en el desarrollo de software.
- Dificultad en la comprensión de sistemas de gran complejidad.
- Limitaciones en los lenguajes de programación.
- Miedo a la introducción de errores por parte de los programadores por el desarrollo de código fuente nuevo.
- Reestructuración del código fuente con limitaciones de tiempo.
- Utilización de soluciones similares en distintos proyectos de software.
- Utilización de plantillas estructurales y funcionales para dar formato al código fuente (algunos paradigmas de programación recomiendan el uso de estas).

Si bien la reutilización de código fuente puede traer ciertas ventajas con respecto a la rápida adaptación del sistema ante los cambiantes requerimientos de los usuarios, en realidad puede implicar ciertas desventajas que ocasionen efectos negativos en el proceso de desarrollo de software, entre los que se destacan:

- Altos costos de mantenimiento.
- Propagación de errores.
- Impacto negativo en el diseño del sistema ya que fomenta el uso de malas prácticas de diseño.
- Incremento innecesario del tamaño del sistema lo que implica una degradación en la performance y calidad del mismo.

Baker (1995) afirma en sus estudios que entre el 20 y 30% del código fuente de grandes sistemas de software corresponde a código fuente “clonado” o reutilizado, sin embargo otros autores (Ducasse et al. (1999), Mayrand et al. (1996), Kontogiannis et al. (1996) y Lague et al. (1997)) sugieren que este porcentaje puede oscilar entre el 20% y 50%.

La identificación de similitudes de código puede servir para varios propósitos (Smith y Horwitz, 2009), entre los que se puede mencionar el estudio de la evolución del código fuente de un proyecto, detección de prácticas de plagio, detección de prácticas de reutilización, extracción de un fragmento de código para “refactorización” del mismo y detección y seguimiento de defectos, fallas y/o virus para su corrección.

Si bien la problemática de detección de este tipo de prácticas es bastante compleja, existen diversas técnicas enfocadas a su detección automática que permiten la detección de fragmentos de código fuente similares (usualmente llamados Clones de Código o Code Clones) pero son fuertemente dependientes del material de entrenamiento, del lenguaje de programación, del tipo de código (algorítmico, definición de interfaces, código de acceso a datos, etc) y del tipo de clon que se pretende detectar.

Roy y Cordy (2009) realizan una clasificación de los tipos de clones (código reutilizado), según las modificaciones que conlleva, y es la más utilizada actualmente en el área:

Tipo-1: Dos fragmentos de código son copias exactas, excepto por los espacios en blanco, tabulaciones y comentarios.

Tipo-2: Dos fragmentos de código son similares, excepto por el nombre de las variables, tipos, literales y funciones.

Tipo-3: Dos fragmentos de código son similares, pero con modificaciones de instrucciones (agregado y/o borrado) y se utilizan diferentes identificadores, tipos, espacios en blanco, comentarios, etc.

Tipo-4: Dos fragmentos de código son semánticamente similares pero su sintaxis es diferente.

Según lo planteado por Ain et al. 2019, existen distintas aproximaciones orientadas a técnicas para detección de tipos de clones en función de la forma en que se realiza la extracción de los datos y el proceso de síntesis cuando se realiza la detección del código clonado. A continuación de describirán brevemente las categorías que las agrupan:

Aproximaciones Textuales: las técnicas de esta categoría utilizan aproximaciones basadas en información textual del código fuente aplicando ligeras transformaciones al código fuente midiendo la similitud en base a la comparación

de secuencias de texto, por lo que están limitadas en su capacidad para reconocer dos fragmentos como un par de clones y detectan más efectivamente los clones de tipo 1 (Vislavski et al., 2018), aunque algunas aproximaciones pueden identificar clones del tipo 2 y 3 (Saini y Singh, 2018).

Aproximaciones Léxicas: Las técnicas incluidas en esta categoría utilizan aproximaciones basadas en tokens (unidad mínima de secuencia de caracteres léxicamente significativa) comparando las subsecuencias para la detección de clones que son capaces de operar a un nivel más alto de abstracción e identificar clones de tipo 1, 2 y 3, pero más eficientemente los de tipo 2 (Vislavski et al., 2018) (Ami y Haga, 2017).

Aproximaciones basadas en árboles: Estas técnicas utilizan árboles sintácticos (Abstract Syntax Tree - AST) que representan el código fuente y miden la similitud de subárboles para la detección de clones. Suelen ser más efectivas para la detección de clones de tipo 3 (Vislavski et al., 2018) aunque también son capaces de detectar todo tipo de clones (Saini y Singh, 2018). Sin embargo, requieren mayor poder de cómputo, lo cual las hace más lentas y costosas (Ami y Haga, 2017).

Aproximaciones basadas en métricas: Estas aproximaciones utilizan métricas que se extraen del código fuente y se comparan para evaluar la similitud, siendo más efectivas para la detección de clones de tipo 3 (Vislavski et al., 2018) aunque también son capaces de detectar los restantes tipos de clones (Saini y Singh, 2018).

Aproximaciones semánticas: Las técnicas incluidas en esta categoría representan el código fuente un grafo de dependencias del programa (PDG) en el que los nodos representan expresiones y sentencias, y los arcos representan dependencias de control y datos. Estas aproximaciones son similares, en cuanto a su capacidad de detección de clones, a las aproximaciones basadas en árboles y en métricas pero son utilizadas principalmente para la detección de clones de tipo 4 (Vislavski et al., 2018).

Aproximaciones híbridas: las técnicas agrupadas en esta categoría usualmente realizan combinaciones de las aproximaciones mencionadas anteriormente, algunas de esas combinaciones se mencionan en Rattan et al. (2013) y en Ami y Haga (2017). Con respecto a capacidad de éstas técnicas para detección de tipos de clones, varía mucho del tipo de combinación que se realice, pero existen combinaciones que son capaces de reconocer los cuatro tipos de clones, algunas más eficientemente que otras.

En el presente trabajo, se propone desarrollar un sistema que permita integrar una combinación de diversas aproximaciones y proponer un nuevo enfoque basado en la implementación de técnicas que son comúnmente utilizadas en el Procesamiento del Lenguaje Natural pero adaptándolas para el procesamiento de textos en formatos estructurados (códigos fuentes en algún lenguaje de programación) para determinar la similitud entre diferentes programas desde el punto de vista léxico, sintáctico y semántico.

Asimismo, la utilización de técnicas de aprendizaje automático posibilitará crear un modelo de detección que permita evaluar la similitud de un archivo fuente contra un conjunto de archivos de código fuente.

Dentro de este contexto, se propone la construcción de un sistema, empleando de técnicas de aprendizaje automático supervisado, tales como Redes Neuronales Artificiales (RNA) y Máquinas de Vectores de Soporte (SVM-Support Vector Machines), comúnmente utilizadas para minería de datos (Jadon, 2016). Dichas técnicas serán realimentadas con métricas de código fuente orientadas al procesamiento de lenguaje natural (Flores Sàez, 2017), con el objetivo de poder determinar un valor indicativo del nivel de similitud de un código fuente con respecto a un corpus, de manera tal de brindarle al usuario una medida cuantitativa que lo ayude en la identificación de fragmentos de reutilización de código.

Metodología

Desde el punto de vista metodológico, para el desarrollo de la tesis propuesta se pueden observar cinco etapas claramente diferenciadas: Creación del Corpus, Pre-procesamiento, Entrenamiento, Producción, y Prueba.

Etapa de Creación de Corpus (material de entrenamiento)

Esta etapa consiste en la creación de material de entrenamiento sobre el cual se aplicarán las técnicas de Aprendizaje Automático Supervisado. Las actividades que se realizan como parte de esta etapa son: generación, tabulación, ordenamiento y etiquetado de los pares de entrenamiento.

Etapa de Pre-procesamiento

Esta etapa consiste en la aplicación de múltiples técnicas de lingüística computacional tales como detección de términos relevantes, aplicación de lexers, parsers, etc, sobre el corpus de entrada para construir/generar adecuadamente las features necesarias para el entrenamiento.

Etapa de Entrenamiento

Esta etapa consiste en partir de la información del Corpus y tomarla como Trainning Set (conjunto de entrenamiento) para el sistema. Básicamente, el sistema tiene la capacidad de aprender en base a ejemplos. Se construye un modelo en base a los ejemplos de entrenamiento y se buscan relaciones o patrones existentes entre los mismos.

Etapa de Producción

Una vez definido el modelo, el mismo podrá tomar como entrada dos documentos y determinar si hay o no similitud entre los mismos.

En la figura 1 se plantea el proceso que utilizará el sistema de detección de similitudes en códigos fuentes expresados en los lenguajes de programación JAVA y Python, en base al corpus elaborado especialmente para el desarrollo del presente trabajo.

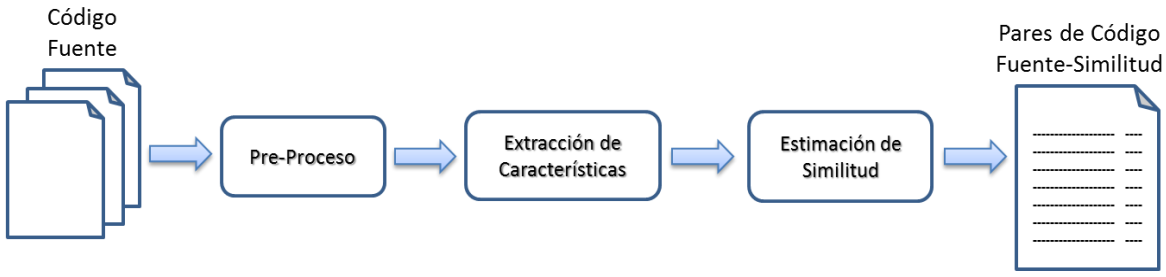

Figura 1: proceso de detección de similitudes en códigos fuente.

Se puede observar que, a partir del conjunto de documentos de entrada (archivos de código), se realiza el pre-proceso que consiste en dar formato al documento, su tabulación y eliminación de información redundante y/o superflua. Con el documento pre-procesado se realiza la extracción de las features relevantes, tales como las métricas basadas en técnicas de Reconocimiento de Implicación Textual de lenguaje natural propuestas por Castillo (2015), adaptadas para su utilización con texto estructurado.

Una vez realizada la estimación de similitud, se generará un archivo final con los resultados obtenidos y se los analizará en base a métricas previamente definidas, y a una revisión manual de expertos humanos.

Etapa de Prueba

Una manera de probar la efectividad el sistema será en base a la medida de Acc (acurracy) que indica la cantidad de veces que el sistema acierta, en función de la cantidad de veces que el sistema realiza una clasificación. Se analizarán y estudiarán diversas métricas para la evaluación de los resultados propios del área de investigación.

Resultados

El trabajo de tesis, que comenzó su desarrollo como parte del proyecto “Modelado para el procesamiento de textos estructurados” con código de identificación UTN4518, se encuentra en desarrollo dentro del proyecto homologado por la Secretaría de Ciencia y Tecnología de la Universidad Tecnológica Nacional, “Modelo para el procesamiento de textos estructurados Fase 2 “ con código de identificación SIECACO0008518. Actualmente esta tesis, se encuentra en la etapa de recopilación, tabulación y estudio de la documentación nueva y revisión de la existente en el marco el proyecto de investigación dentro del cual se enmarca. Esta actividad es fundamental para explicar las aportaciones al conocimiento que realiza la tesis al estado del conocimiento actual y como parte de esta actividad también se realiza el estudio y adaptación del sistema de Reconocimiento de Implicación Textual (RTE) Sagan propuesto por Castillo (2015) para la detección de similitudes en código fuente.

Además, se está realizando la recopilación, análisis y tabulación de los datos de entrenamiento del sistema propuesto para poder integrarlos con la arquitectura que define el comportamiento global del sistema.

Conclusiones

El trabajo realizado hasta el momento se ha centrado en el relevamiento de los datos necesarios para la elaboración del material de entrenamiento del sistema, diseño preliminar de la arquitectura del sistema y estudio del estado del arte de la temática planteada. Se prevee en trabajos futuros, profundizar en el mejoramiento de los datos de entrenamiento y refinamiento de la arquitectura, como así también el desarrollo del sistema de detección de similitudes en código fuente y definición del modelo subyascente, para lo cual se realizarán los cálculos necesarios para determinar las features definidas como datos de entrada para el mismo mediante la aplicación de múltiples técnicas de lingüística computacional tales como detección de términos relevantes, aplicación de lexers, parsers, etc, sobre el corpus con el objetivo de construir/generar adecuadamente las features necesarias para el entrenamiento.

Adicionalmente se estudiarán los diferentes mecanismos y algoritmos de visualización de diferencias en archivos de formato estructurado, y diferentes algoritmos para tratar con grandes volúmenes de información.

Referencias

Ain, Q., Butt, W., Anwar, M., Azam, F., & Maqbool, B. (2019). A Systematic Review on Code Clone Detection, in IEEE Access, vol. 7, pp. 86121-86144.

Ami, R., & Haga, H. (2017). Code Clone Detection Method Based on the Combination of Tree-Based and Token-Based Methods. Journal of Software Engineering and Applications, 10, 891-906.

Baker, M. (1995). On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), pp. 86-95, Toronto, Ontario, Canada.

Castillo, J. (2015). Tesis doctoral: Reconocimiento de Implicación Textual y Aplicaciones. Lugar: FaMAF-UNC, Ciudad: Córdoba, País: Argentina. Idioma: Español. Publicación impresa, biblioteca de FaMAF, UNC.

Ducasse, S., Rieger, M., & Demeyer, S. (1999). A language independent approach for detecting duplicated code, in Proc. IEEE Int. Conf. Softw. Maintenance (ICSM), pp. 109–118.

Jadon, S. (2016). Code clones detection using machine learning technique: Support vector machine. 2016 International Conference on Computing, Communication and Automation (ICCCA). IEEE. Noida, India.

Flores Sàez, E. (2017). Detección de reutilización de código fuente monolingüe y translingüe. Procesamiento del Lenguaje Natural. pp. 163-166.

Fowler, M., Beck, K., Brant, T., Opdyke, W., & Roberts, D. (1999). Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman.

Kontogiannis, K., Mori, R.D., Merlo, E., Galler, M., & Bernstein, M. (1996). Pattern matching for clone and concept detection. J. Autom. Softw. Eng. 3(1), pp. 79–108.

Lague, B., Proulx, D., Mayrand, J., Merlo, E.J., & Hudepohl, J. (1997). Assessing the benefits ofincorporating function clone detection in a development process. In: Proceedings of 1stIEEE International Conference on Software Maintenance, Washington, DC, USA, pp. 314–321.

Mayrand, J., Leblanc, C. & Merlo, E. (1996). Experiment on the automatic detection of functionclones in a software system using metrics. In: Proceedings of 1st IEEE InternationalConference on Software Maintenance, Monterey, CA, pp. 244–254.

Saini, N., & Singh, S. (2018). Code clones: Detection and management, Procedia Comput. Sci., vol. 132, pp. 718–727.

Shafieian, S., & Zou, Y. (2012). Comparison of clone detection techniques. Technical report, Queen.

Smith, R., & Horwitz, S. (2009). Detecting and Measuring Similarity in Code Clones. International Workshop on Software Clones (IWSC'09), pp. 28-34.

Rattan, D., Bhatia, R. & Singh, M. (2013). Software clone detection: A systematic review. Elsevier. Information and Software Technology 55, pp. 1165–1199.

Resnik, P. (1995). Using information content to evaluate semantic similarity. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, pp. 448–453.

Roy, C., & Cordy, J. (2007). A survey on software clone detection research, Queen's School Comput., Tech. Rep. 541, vol. 115, pp. 64–68.

Roy,C., Cordy, J., & Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, Sci. Comput. Program., vol. 74, pp. 470-495.

Vislavski, T., Rakić, G., Cardozo, N., & Budimac, Z. (2018) LICCA: A tool for cross-language clone detection, in Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER), pp. 512–516.