ISSN 0361-7688, Programming and Computer Software, 2017, Vol. 43, No. 1, pp. 47–50. © Pleiades Publishing, Ltd., 2017.
Original Russian Text © G. Sidorov, M. Ibarra Romero, I. Markov, R. Guzman-Cabrera, L. Chanona-Hernández, F. Velásquez, 2017, published in Programmirovanie, 2017, Vol. 43, No. 1.

Measuring Similarity Between Karel Programs Using Character and Word N-Grams[1]

G. Sidorov[a*], M. Ibarra Romero[a], I. Markov[b**], R. Guzman-Cabrera[c***], L. Chanona-Hernández[d****], and F. Velásquez[e*****]

[a] Instituto Politécnico Nacional (IPN), Center for Computing Research (CIC), Mexico City, Mexico
[b] Instituto Politécnico Nacional (IPN), Center for Computing Research (CIC), Mexico City, Mexico
[c] University of Guanajuato, Campus Irapuato-Salamanca, Engineering Division, Mexico
[d] Instituto Politécnico Nacional, School of Mechanical and Electrical Engineering (ESIME), Mexico City, Mexico
[e] Polytechnic University of Queretaro, Mexico

E-mail: *sidorov@cic.ipn.mx, **markovilya@yahoo.com, ***guzmanc81@gmail.com, ****lchanona@gmail.com, *****francisco.castillo@upq.mx

Received August 5, 2016

We present a method for measuring similarity between source codes. We approach this task from the machine learning perspective using character and word *n*-grams as features and examining different machine learning algorithms. Furthermore, we explore the contribution of the latent semantic analysis in this task. We developed a corpus in order to evaluate the proposed approach. The corpus consists of around 10,000 source codes written in the Karel programming language to solve 100 different tasks. The results show that the highest classification accuracy is achieved when using Support Vector Machines classifier, applying the latent semantic analysis, and selecting as features trigrams of words.

Keywords: machine learning, similarity, Karel programming language, character *n*-grams, word *n*-grams, SVM, LSA

DOI: 10.1134/S0361768817010066

1. INTRODUCTION

The motivation of this work is to support the professors involved in the task of teaching programming techniques, when a frequent problem is the correction of tasks and/or programming tests performed by the students, since the reviews of exercises are often performed incompletely and/or inequitably under time constraints.

During the introductory programming courses it is common to use auxiliary tools and educational programming languages to provide an accessible understanding of the programming basics. A popular tool in this environment is Karel the Robot [1] developed by Richard E. Pattis and its corresponding programming language, which is used, for example, in the Mexican Olympiad in Informatics1, where we have obtained the codes used in the present work.

The main problem addressed in this paper is the following: how to train a classifier on a training data and how to properly classify a program that has not been presented in the training set by the proposed solution or by the problem that is being solved.

In order to tackle this task, the degree of similarity between programs in Karel language is determined. The analysis of the similarities helps to identify the following aspects of code reuse: (1) when additional information reveals a similarity between two programs, one may detect evidence of plagiarism and/or of sharing a program code; an exchange of ideas/solutions between students; or whether professors instruct their students in similar programming techniques; (2) when the programs tend to be different, it is possible to determine the different methods employed for solving a given problem and its originality.

The rest of the paper is structured as follows. Section 2 describes related work on similarity detection. Section 3 explains with some detail how the test dataset was formed. Section 4 describes the construction of the vector space model (selection of features and their values). Section 5 presents the pre-processing stage. Section 6 presents and discusses the results. Section 7 draws the conclusions from this work and points to the possible directions of future work.

[1]The article is published in the original.

2. RELATED WORK

Some of the works related to similarity detection are focused on plagiarism; however, in this paper, we are concerned with similarity as such, since plagiarism refers to a situation of fraud, which is not always possible to detect, especially automatically. Therefore, although we make references to works on plagiarism, in this paper, we use the term "similarity between programs".

We start with the premise that two programs are similar if one can be converted into other by using a small number of transformations. The transformations can range from simple changes in comments to modifications in the control logic. Therefore, if programs have a lot of code in common, the degree of similarity between them is higher, since a smaller number of transformations is required.

The early similarity or plagiarism detection programs appeared in the 70's. The feature counting techniques (also called metrics) were used as in Halstead [2] and McCabe [3]. The main idea of this approach is to keep statistics of characters in the source program (called correlation of characters) in order to make a comparison with all the programs and, finally, perform a summary ordered from highest to lowest values based on the statistical features.

In the 80's, a combination of metrics and chain comparison algorithms was used (a technique of converting different elements of a program into attributes that helps its identification). The Plague[2] and Wise [4] programs are representatives of this period. These works consisted in ignoring the comments, blank lines, spaces, and tabs and making use of a chain comparing program (like diff, grep, etc.), which is based on a line-by-line comparison using the Levenshtein distance (also known as the edit distance, i.e., the number of operations required to convert one string into another).

Tran and Gitchell [5] in the 90's, worked on the basis of the analysis of trees generated by a program. Under this approach, the tree representation is reduced by using standard lexical analysis; tree chains are aligned in order to obtain a greater subsequence, like a degree of similarity.

In recent works [6], the latent semantic analysis (LSA) is applied in order to reduce the number of dimensions of the vector space that represents a program. These works are usually based on the cosine similarity. The cosine similarity technique consists in creating a vector representation based on normalized frequency of character n-grams (usually bigrams or trigrams) in each program, and relying on these representations, to determine how similar the programs are.

Recently, the idea of linking the semantics of the source code with the documentation, definition of variables, names of procedures and/or comments has emerged in order to improve the understanding of the data on the requirements and design of codes sources [7]. One example is the Moss (Measure for Software Similarity)[3] program that was developed at Stanford University [8]. This software is based on the Winnowing algorithm that divides a program into continuous chains called k-grams. Each k-gram is converted into a hash, and the subset of all k-gram hashes represents the documents' fingerprints. Subsequently, two programs are compared according to the fingerprints similarity.

Another example is the Plaggie[4] software that was developed at the Technical University of Helsinki. This application creates strings of characters and uses the Running Karp Rabin Matching algorithm, which also extracts the documents fingerprints by using hash functions of a special type, and then their comparison is performed.

3. CORPUS

The corpus (large set of data) was built from the website that contains the Karel (Karelotitlan[5]) language programs in order to measure the similarity between the source codes. This website is a domain where young people, mostly under 18, learn how to program using the Karel language. The selection of codes that form the corpus was done applying simple random sampling.

The corpus "Karel_100" contains 9,341 programs that 100 students wrote to solve different tasks. There are 100 tasks and approximately 100 programs per task. Note that, in this case, the programs are classified only by task. Programs within the website Karelotitlan are precisely grouped by a person/student and a task, so in this work, we focus on 100 different tasks. The corpus is freely available on our website[6].

4. CONSTRUCTION OF THE VECTOR SPACE MODEL

In order to build the vector representation of the documents, first, the combinations of contiguous terms are to be extracted. These sequences are called n-grams. The value of n indicates the number of elements in an n-gram. For example, there are bigrams, trigrams, 4-grams, 5-grams, etc. N-grams allow obtaining very good results in various machine learning tasks. There is also an option of using syntactic n-grams [9], graph based approach [10], and tree edit distance [11]; however, in this study, we are focusing on traditional n-grams, since we are working with source codes. The detailed description of the n-grams and the vector space model concepts are presented, for example, in [12].

[2]http://plague-inc-evolved.softonic.com

[3]https://theory.stanford.edu/aiken/moss/

[4]https://www.cs.hut.fi/Software/Plaggie/

[5]http://www.cmirg.com/Karelotitlan/

[6]http://www.cic.ipn.mx/~sidorov/

Additionally, instead of using words and/or n-grams of words, one can also use n-grams of characters according to their appearance in a program, which is also common in natural language processing, for example, in the authorship attribution task. Each word (or n-gram of words or n-gram of characters) in our representation is a dimension in the vector space model.

In order to assign a numerical value to each element in the vector space, one of the following values is used:

tf (term frequency), $tf_{t,d}$ means frequency of the term t in the document d, that is, the number of times a term appears in a document. The idea behind this value is very obvious: the more times a term appears in a document, the more important this term is for this document.

idf (inverse document frequency), this value implies that a term is weighted according to the number of documents in the corpus in which it appears: how common a term t in the collection of documents idf_t. The idea behind this value is that if a term appears in fewer documents, it is more valuable, since it makes it easier to distinguish between the documents.

Finally, $tf - idf_{t,d} = tf_{t,d} \times idf_t$ is used. This value is high when a term appears many times in a small number of documents; the value is low when a term appears only a few times in a document or when it appears in many documents, which does not allow using this term to distinguish well between the documents. There can be the case when two documents have similar content but their terms counts are different. This situation may occur when one document is very large and the other is very small, or when one document describes the same process several times and the other only once. Therefore, it is useful to normalize the weightings of the document terms (tf) in relation to the size of the document.

In our case, the Euclidean normalization is used, which assigns a value between 0 and 1 to each term according to the number of appearances in the document [13]. The Euclidean norm is the square root of the sum of the squares of all elements of a vector in the vector space model.

5. PRE-PROCESSING STAGE

Pre-processing is performed before processing the text. Firstly, we removed comments, since they do not represent source code information, and furthermore, they are easy to modify in case one want to conceal the code reuse. Moreover, we removed numerical terms, which is justified by the fact that in our case not taking numerical terms into account gives better results.

In order to detect the similarity between the codes, it is necessary to build a vector representation of the source codes (considered as text). Firstly, we made a list that contains all words that are present in the codes and their corresponding frequencies. For our experiments, we considered only those words, whose frequency is greater than or equal to two. The unique terms (frequency equals one) were omitted in order to reduce the number of dimensions of the vector space and achieve a lower computational cost.

Also the following syntactical terms were omitted: “(", ")”, “;”. Additionally, we eliminated the terms without lexical value known as “stop words”. These are most commonly used terms, which normally do not contribute to the lexical meaning, reflecting only grammatical function, for example, articles (“the”, “a”, “an”), prepositions (“to”, “with”, etc.), conjunctions, etc. In this sense, after performing the analysis of the keywords of the Karel language, the following words were omitted: “then”, “times”, “do”, “how”, “defined-new-instruction”, “beginning-of-program”, and “end-of-program”.

Once we have constructed the vector representation in terms of tf and/or $tf - idf$, the latent semantic analysis (LSA) [14] is applied in order to find “latent relations” and to reduce the number of dimensions of the vector space. In our experiments, LSAis used to reduce the number of dimensions to 50, 100, 150, 200, 300, 400, 500, and 600. In general, LSA consists in applying singular value decomposition to the vectors.

6. EXPERIMENTAL RESULTS AND DISCUSSION

The experiments were carried out using the freely available machine learning software, WEKA [15]. Two machine learning algorithms were examined: (1) Naive Bayes, which is a probabilistic learning algorithm based on Bayes’ theorem and (2) SVM (support vector machines), this machine learning algorithm makes the separation between classes by means of the optimal hyperplanes.

To perform the classification of the results we used the method called k-fold cross-validation with the parameter k equal to 10. The method consists in dividing the corpus into k subsets. One of the k subsets is used for testing, and the other $k - 1$ subsets are used for training. The procedure is repeated k times; each time different k subset is used for testing. Then the result is computed as the average across all k trials.

The best result was obtained with the SVM algorithm (69.9%), using trigrams of words, $tf - idf$ weighing, and reducing the number of dimensions to 500 using LSA. By reducing the number of dimensions, we achieved better results (curiously, 500 is the “magic” number that most authors acknowledge as the most appropriate of the LSA). Without reducing the number of dimensions, the algorithm obtains 68.5%, which is 1.4% worse. Note that the baseline (we used words as features with the $tf - idf$ weightings without applying the LSA) obtains the best result of 43.25%, which was surpassed by more than 25%.

The obtained results are very promising, considering the fact that it would take a long time for a human annotator to perform the task, and it would be a tedious job to review thousands of source codes trying to reveal the similarity between them. Moreover, performing this task manually would certainly lead to a high probability of making a mistake. It can be even considered that with a sufficiently large set of source codes, it is almost impossible to accomplish the task manually, while the proposed solution compares thousands of codes in a few minutes.

7. CONCLUSIONS AND FUTURE WORK

In this section, we draw the conclusions from this work and indicate the possible directions of future work.

• The main goal of this work consisted in measuring similarity between programs using the vector space model and machine learning methods. The task is useful, for example, for detecting code reuse.

• A large corpus of the Karel programming language codes was built, and it is freely available for use and testing.

• The experiments were conducted on the developed corpus using the WEKA tool. The results showed that the representation of programs with *n*-grams of words or *n*-grams of characters yields better results on similarity detection than using only words as features.

• We also showed that the application of the latent semantic analysis further improves the experimental results.

The directions of future work are the following:

• To apply soft similarity [16], which allows taking into account the similarity between features in a vector space model. The application of soft similarity already proved to give better results in some natural language processing tasks.

• Even though in the formal language used in the programs development, each element is essential to determine the correct dimensionality, it is important to note that heuristics are needed to improve the similarity calculation. A possible direction of future work would be to combine similar metrics to those of the 70's and 90's with the vector representation of programs.

• To implement a process similar to morphological analysis of natural language to treat identifiers, roots, lemmas, etc. This would allow unifying inflections of terms, for example, if we have several terms such as "calculate", "calculation" "calculating", we have to keep in mind that they refer to the same concept, and a similar situation may occur with their synonyms, which may be the words "computing" and "counting".

• To continue modifying the number of dimensions in the representation of the programs with different LSA parameters.

REFERENCES

1. R. E. Pattis, J. Reoberts, and M. Stehlik, *Karel the Robot: Gentle Introduction to the Art of Programming*, 2nd Ed. (John Wiley Sons, 1994).
2. M. H. Halstead, *Elements of Software Science* (North Holland, New York, 1977).
3. T. J. McCabe, "A complexity measure", IEEE Trans. Software Eng. **2**(4), 308–320 (1976).
4. M. J. Wise, "YAP: Improved detection of similarities in computer program and other texts", in *Proceedings of SIGCSE'96 Technical Symposium* (Philadelphia, USA, 1996), pp. 130–134.
5. N. Tran and D. Gitchell, "Sim: A utility for detecting similarity in computer programs", SIGCSE Bull. **31**(1), 266–270 (1999).
6. G. Cosma, "An approach to source-code plagiarism detection and investigation using latent semantic analysis", PhD Dissertation (Department of Computer Science, University of Warwick, 2008).
7. S. K. Hsu and S. J. Lin, "A block-structures model for source code retrieval," in *Proceedings of Intelligent Information and Database Systems, Third International Conference, ACIIDS 2011*, 2011, pp. 161–171.
8. S. Saul, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting", in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (ACM, New York, NY, USA, 2003), pp. 76–85.
9. J. P. Posadas-Durán, I. Markov, H. Gómez-Adorno, G. Sidorov, I. Batyrshin, A. Gelbukh, and O. Pichardo-Lagunas, "Syntactic N-grams as features for the author profiling task", in *Conference and Labs of the Evaluation Forum, Working Notes of CLEF* 2015 (Toulouse, France, 2015), **vol. 1391**.
10. H. Gómez-Adorno, G. Sidorov, D. Pinto, and I. Markov, "A graph based authorship identification approach", in *Conference and Labs of the Evaluation Forum, Working Notes of CLEF* 2015 (Toulouse, France, 2015), **vol. 1391**.
11. G. Sidorov, H. Gómez-Adorno, I. Markov, D. Pinto, and N. Loya, "Computing text similarity using tree edit distance", in *Proceedings of the Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC), 2015 Annual Conference of the North American* (Redmond, WA, USA, 2015), pp. 1–4.
12. G. Sidorov, "Should syntactic N-grams contain names of syntactic relations?", Int. J. Computational Linguistics Appl. **5**(1), 139–158 (2014).
13. *Information Retrieval* (Cambridge University Press, New York, NY, 2008).
14. S. Deerwester, S. T. Dumais, G. W. Furnas, and T. K. Landauer, "Indexing by latent semantic analysis", J. Am. Soc. Inform. Sci. **41**(6), 391–407 (1990).
15. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update", SIGKDD Explorations **11**(1) (2009).
16. G. Sidorov, A. Gelbukh, H. Gómez-Adorno, and D. Pinto, "Soft similarity and soft cosine measure: Similarity of features in vector space model", Computación y Sistemas **18**(3), 491–504 (2014).