Personality Recognition Applying Machine Learning Techniques on Source Code Metrics

Hugo A. Castellanos
Universidad Nacional de Colombia
Bogotá, Colombia
hacastellanosm@unal.edu.co

ABSTRACT

Source code has become a data source of interest in the recent years. In the software industry is common the extraction of source code metrics, mainly for quality assurance purposes. In this paper source code metrics are used to consolidate programmers profiles with the purpose to identify different personality traits using machine learning algorithms. This work was done as part of the Personality Recognition in SOurce COde (PR-SOCO) shared task in the Forum for Information Retrieval Evaluation 2016 (FIRE 2016).

CCS Concepts

•Information systems → Content analysis and feature selection; •Computing methodologies → *Supervised learning by regression;* Cluster analysis; **•General and reference → Metrics; •Software and its engineering** → Parsers;

Keywords

Personality recognition; Source code metrics; Support Vector Regression

1. INTRODUCTION

Pieces of text have been always of interest in information retrieval as text based documents contain valuable information about the author. During recent decades source code has become a source of valuable information as well. Many efforts in this field have been addressed to improve both processes and products in the software development industry [8].

The main efforts in source code analysis have been focused in forensics applications like author recognition [5], and plagiarism detection [2]. Several techniques have been used successfully in the mentioned tasks like n-grams, source code metrics, coding styles and abstract syntax trees [6]. Other applications of source code analysis include feature location [3], topics identification [7], among others.

The PR-SOCO shared task consisted in predict the personality traits of a programmer given a set of his/her source codes. These source codes as any other production of a human being may be influenced by personality.

In this work, the use of source code metrics is proposed to find information about the program author. Specifically, the author personality traits based on the Big-5 personality test. In addition, machine learning methods are used to predict the personality traits based on the extracted source code metrics.

The rest of this paper is organized as follows. Section 2 presents a general background on source code metrics. Section 3 describes the proposed approach. Section 4 presents the machine learning strategies. Section 5 presents the obtained results. Finally, Section 6 concludes the paper.

2. BACKGROUND ON SOURCE CODE METRICS

According to Malhotra [8], software metrics are used to *assess the quality of the product or process used to build it.* Such metrics have the following characteristics:

- Quantitative: metrics have a value.
- Understandable: the way the metric is calculated must be easy to understand.
- Validatable: metrics must capture the attributes which they were designed to.
- Economical: it must be economical to capture the metric.
- Repeatable: if measured several times the results should be the same.
- Language independent: the metrics should not depend to a specific language.
- Applicability: the metric should be applicable in any phase of the software development.
- Comparable: the metric should correlate with another metric capturing the same concept.

Source code metrics must have a scale which can be:

- Interval: it is given by a defined range of values.
- Ratio: it is a value which has an absolute minimum or zero point.
- Absolute: it is a simple count of the elements of interest.
- Nominal: it is a value which mainly defines a discrete scale of values, like 1-present or 0-not present.
- Ordinal: it is a categorization which is intended to order or rank, for instance levels of severity: critical, high, medium, etc.

The metrics can be classified according the intended measure:

- Size: usually intended to estimate cost and effort. The most popular metric in this category are the source lines of code (SLOC). But in object oriented languages the size can be measured by the number of classes, methods and attributes.
- Software quality: intended to measure the quality of the software, this metric can be divided in the following categories:
  - Based on defects: they consist in measure the level of defects. The main metrics of this category are: the defect density defined as the number of defects by SLOC; defect removal effectiveness which is defined as the number of defects removed in a phase divided by latent defects. If the latent defects are unknown then can be estimated based on previous phases.
  - Usability: this kind of metrics are intended to measure the user satisfaction using the software. The satisfaction can be given be the ease to use and learn.
  - Complexity metrics [9]: they are oriented to produce a measure on the difficulty to test or maintain a piece of source code. This metric also give information about the amount of instructions during execution.
  - Testing: intended to measure the progress of testing over a software
- Object oriented metrics: intended to measure object oriented paradigm features. They can be divided in:
  - Coupling: measure of the level of interdependence between classes, it is calculated counting the number of classes called by another class.
  - Cohesion: measures how many elements of a class are functionally related to each other.
  - Inheritance: it measures the depth of the class hierarchy.
  - Reuse: measure of the amount of times that a class is reused.
  - Size: intended to measure the size but not only in lines of code but also in the particularities of object oriented paradigm, like method count, attribute count, class count, etc.
- Evolutionary metrics: try to measure the evolution of a software based on different elements like revisions, refactorings, bug-fixes. The measure how much lines of code are new, modified or deleted.

Additionally the empirical Halstead metrics [4] should also be considered. The base to calculate these metrics are the operands (identifiers) and operators (keywords, ++, +). Equation 1 consist in the sum of the unique operators (n_1) and operands (n_2). *Length*, described in Equation 2, is the sum of the total number of operands (N_1) and operators (N_2).

$$n = n_1 + n_2 \tag{1}$$

$$N = N_1 + N_2 \tag{2}$$

The Halstead *volume* (V), described in Equation 3, is a measure of size but it is also interpreted as the number of mental comparisons that were needed to write a program with length N. Moreover, the *difficulty* (D), shown in Equation 4, describes the difficulty to write a program. It is highly related to *volume* because as it increases the *difficulty* also does.

$$V = N \log_2 n \tag{3}$$

$$D = \frac{n_1}{2} \cdot \frac{N_2}{n_2} \tag{4}$$

The *effort* (E) described in Equation 5, indicates the effort required to write a program of high difficulty.

$$E = D \cdot V \tag{5}$$

Finally, the *effort* is the base to calculate the *time to understand/implement* (T) and *bugs delivered* (B), as can be seen in Equations 6 and 7, respectively. The *time* metric is related to the Stroud number [12], which is the *"number of elementary discrimination per second"*. Stroud claimed that this number ranges from 5 to 20, but the Halstead's experiments indicated empirically that the best number in this case was 18.

$$T = \frac{E}{18} \tag{6}$$

$$B = \frac{E^{\frac{2}{3}}}{3000} \tag{7}$$

3. SOURCE CODE ANALYSIS FOR PERSONALITY RECOGNITION

Text documents, contains information about the author. In the work described in [1], the authors were able to show that certain personality traits could be predicted based on a text, in this case, an essay.

The present work starts from the hypothesis that source code, as a form of text, leaves traces of the author's personality traits. To the scope of this work source code is a text document written by a single author. It is worth mentioning that a single problem solution could be implemented in several ways by a programmer which give a certain guaranty of uniqueness.

To develop this hypothesis, a method is proposed to extract metrics from source code to be able to predict the personality traits. In Figure 1 the general method is summarized. As first step the source examples provided are separated into individual files. Later a set of metrics is extracted from the source codes using a source code analyzer. With the extracted metrics as an input, machine learning methods are applied in order to predict the personality traits of the authors. Finally, the results are presented.

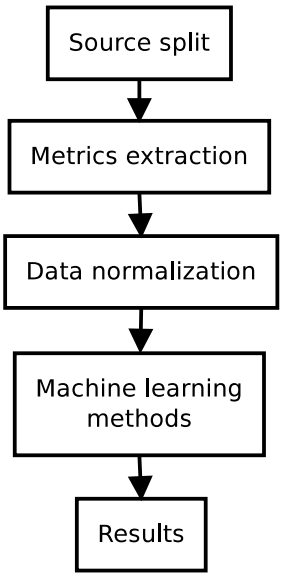

Figure 1: Process summary.

The provided corpus consisted in a source code file per person, and another file which indicates author and his/her personality traits (ground truth). Each source code file contained several source code pieces divided by a mark. The file was split into several individual files keeping track of the author-file relationship.

An analyzer was written using ANTLR 4 [10] with the java grammar. From each individual file the source code metrics described in Table 1 were extracted.

As can be seen most of the metrics are based in counting and obtaining the average. All the metrics were normalized, such normalized data were the input of the machine learning algorithms.

As the extracted metrics are from similar categories, a hierarchical clustering using the Ward's method [13] was applied. It was found that certain related metrics were too close to each other. Therefore, they were consolidated as follows:

- *Length metrics*: contain the metrics related to some length/size measure and it is calculated as the average among: amount of files, average source lines of code, average class number per file, average source code lines per class, average attributes per class, average methods per class, average class name length, and the average number of parameters.

- *Complexity metrics*: contain the metrics related with algorithm complexity and it is calculated as the average of: average amount of for loops, average amount of while loops, average amount of if clauses, average amount of if-else clauses, and the average identifier length.

- *Halstead*: contains all the Halstead metrics extracted, it was calculated as the average of: Halstead bugs delivered, Halstead difficulty, Halstead effort, Halstead time to understand or implement, Halstead volume.

Table 1: Metrics extracted from source code

Metric	**Basic description**
Amount of files	The total amount of files.
Average source lines of code	The average of source lines of code.
Average class number per file	The average of classes per source code file.
Average source code lines per class	The average of source code lines per class.
Average attributes per class	The average of attributes contained in a class.
Average methods per class	The average number of methods contained in a class.
Average class name length	The average length of a class name.
Average amount of for loops	The average amount of for loops contained in a method.
Average amount of while loops	The average amount of while loops contained in a method.
Average amount of if clauses	The average amount of if clauses contained in a method.
Average amount of if-else clauses	The average amount of if-else clauses contained in a method.
Average identifier length	The average identifier length per files.
Average parameters	Average number of parameters in methods.
Average ciclomatic complexity	Indicates the cyclomatic complexity average.
Average of static attributes	The average number of static attributes contained in a class.
Average of static methods	The average of static methods contained in a class.
Halstead bugs delivered	Indicates the number of possible bugs generated based on the halstead metrics.
Halstead Difficulty	An index which measures the difficulty to write the program.
Halstead Effort	An index which measures the necessary effort to write the code.
Halstead Time to understand or implement	An index which indicates the time taken to write a source code.
Halstead volume	Indicates how much information the reader needs to understand the code.

4. MACHINE LEARNING METHODS

In this section the used machine learning methods are described. Each one corresponds to a submission sent to the shared task: submission 1 corresponds to support vector regression (SVR) over source code metrics, submission 2 corresponds to extra trees regressor (ETR), and submission 3 corresponds to support vector regression over averages.

4.1 Support vector regression (SVR) on metrics

A SVR algorithm was used jointly with the extracted metrics as input. For each personality trait an independent SVR was used and a 6-fold cross validation was executed over the corpus. The best parameters according with this validation can be seen in the Table 2. The Figure 2 shows the resulting mean squared error (y axis) versus the gamma variation (x axis) with the best C and ϵ values in logarithmic scale in cross validation. This behavior was similar for all the personality traits.

4.2 Extra trees regressor (ETR) on metrics

Another method applied was the Extra trees regressor, for each personality trait a 6 fold cross validation was performed. For the parameter number of estimators for all traits the best value was 77.

4.3 Support vector regression (SVR) on averages

Based on the clustering results a SVR was used with the metrics averages as input, i.e., length metrics, complexity metrics, and Halstead metrics. The first step was to calculate the variance. As the complexity metrics variance was too low, it was removed and only the length and Halstead average metrics were used as input.

The best parameters according with cross validation can be seen in Table 3. The graphics of γ versus error for the best C and ϵ values have a similar behavior of the one shown in Figure 2.

Table 2: Best parameters with SVR with metrics according with cross validation

Personality trait	C	γ	ϵ
Emotional stability	32	8	2^{-14}
Extroversion	32	16	2^{-12}
Openness to experience	32	16	2^{-19}
Agreeableness	32	8	2^{-10}
Conscientiousness	32	16	2^{-54}

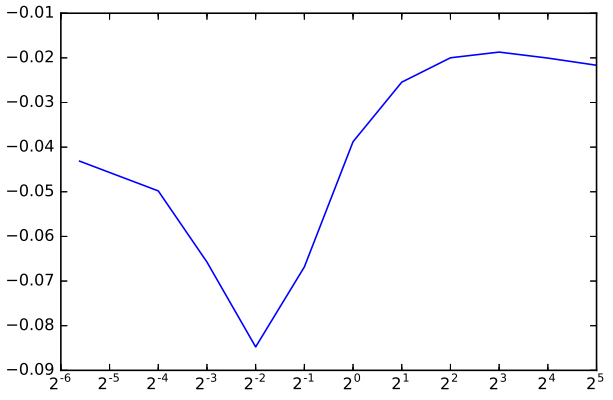

Figure 2: Variation of γ parameter in SVR versus the resulting error in cross validation with the best C and ϵ parameters.

Table 3: Best parameters for SVR with metric averages according with cross validation

Personality trait	C	γ	ϵ
Emotional stability	32	$\frac{1}{49}$	2^{-11}
Extroversion	32	2^{-3}	2^{-10}
Openness to experience	32	$\frac{1}{49}$	2^{-10}
Agreeableness	32	2	2^{-11}
Conscientiousness	32	0.5	2^{-37}

5. RESULTS

Using the mentioned algorithms with the previously described inputs and parameters, the prediction was done on the test dataset. Results can be seen in the Tables 4, 5, and 6.

The three proposed methods obtained a similar performance with Root Mean Squared Error (RMSE). The SVR with metrics have slightly better results. This could be caused by the removal of the complexity metrics.

When evaluated with RMSE the Openness trait was the best result in all the three applied methods, being consistent

Table 4: Results over test data with SVR using metrics as input

Personality trait	MSE	PC
Emotional stability	11.83	0.05
Extroversion	9.54	0.11
Openness to experience	8.14	0.28
Agreeableness	10.48	-0.08
Conscientiousness	8.39	-0.09

Table 5: Results over test data with Extra Tree Regressor using metrics as input

Personality trait	MSE	PC
Emotional stability	10.31	0.02
Extroversion	9.06	0.0
Openness to experience	7.27	0.29
Agreeableness	9.61	-0.11
Conscientiousness	8.47	0.16

Table 6: Results over test data with SVR using metric averages as input

Personality trait	MSE	PC
Emotional stability	10.24	0.03
Extroversion	9.01	0.01
Openness to experience	7.34	0.3
Agreeableness	9.36	0.01
Conscientiousness	9.99	-0.25

with other participant results, and showing better results than the baseline in submissions 2 and 3. Conscientiousness followed with the best error for the SVR and Extra Tree Regressor.

The worst predicted trait with RMSE was Emotional Stability/Neuroticism in all methods, based in the results of other participants[1], this was a general result [11]. A deep study in this particular trait is required to improve the results.

When measured with Pearson Product-Moment Correlation (PC), the results are very different among runs. But submissions 2 and 3 showed much better results compared with baseline because indicates a stronger correlation that the one showed in the baseline. The SVR with averages has an important correlation in openness with value of 0.3 and conscientiousness with value of -0.25. In the ETR run, openness was the highest value with 0.29. SVR over metrics in openness also had the highest value with 0.28. This trait was the most consistent among all the used methods.

It is interesting that PC shows correlations with openness and conscientiousness. This is a good result because indicates that the used metrics have certain relationship whit the mentioned personality traits. Compared with the baseline RMSE, the proposed method performed slightly better, but still it is not significant, which shows that more work is required to obtain a good predictor of personality. Therefore, it is necessary to include more source code metrics within this study. This could lead to find that certain metrics are related to specific personality traits.

6. CONCLUSIONS AND FUTURE WORK

The source code metrics extracted and used as input to the machine learning methods were enough to get a close prediction of several personality traits. Other approaches can be consulted in [**?**] which shows other results and approximations for the PR-SOCO task.

As the PC denotes certain correlation, in this case particularly with openness, this could mean that the metrics considered in this work are likely related to the mentioned trait. However, as there are several other metrics with different purposes, like quality, readability, etc., the use of more of those metrics could improve the prediction. Other metrics not considered in this study may have better relationships with the personality traits. This work could be extended by exploring other metrics an its relationship with each personality trait.

7. REFERENCES

[1] S. Argamon, S. Dhawle, M. Koppel, and J. W. Pennebaker. Lexical predictors of personality type. *Proceedings of joint annual meeting of the interface and The Classification Society of North America*, pages 1–16, 2005.

[2] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing Programmers via Code Stylometry. *USENIX sec*, pages 255–270, 2015.

[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: A taxonomy and survey. *Journal of software: Evolution and Process*, 25(1):53–95, 2013.

[1]http://www.autoritas.es/prsoco/evaluation/

[4] M. H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[5] D. I. Holmes and F. J. Tweedie. Forensic Stylometry: A Review of the {CUSUM} Controversy. *Revue Informatique et Statistique dans les Science Humaines*, pages 19–47, 1995.

[6] R. R. Joshi and R. V. Argiddi. Author Identification : An Approach Based on Style Feature Metrics of Software Source Codes. 4(4):564–568, 2013.

[7] A. Kuhn, S. Ducasse, and T. Gîrba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, 2007.

[8] R. Malhotra. *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. CRC Press, 2015.

[9] T. J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[10] T. Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.

[11] F. Rangel, F. González, F. Restrepo, M. Montes, and P. Rosso. Pan at fire: Overview of the pr-soco track on personality recognition in source code. In *Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016*, CEUR Workshop Proceedings. CEUR-WS.org, 2016.

[12] V. Y. Shen, S. D. Conte, and H. E. Dunsmore. Software science revisited: A critical analysis of the theory and its empirical support. *IEEE Transactions on Software Engineering*, (2):155–165, 1983.

[13] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.