Pisco: A Computational Approach to Predict Personality Types from Java Source Code

Matthias Liebeck
Institute of Computer Science
Heinrich Heine University
Düsseldorf
D-40225 Düsseldorf, Germany
liebeck@cs.uni-duesseldorf.de

Pashutan Modaresi
Institute of Computer Science
Heinrich Heine University
Düsseldorf
D-40225 Düsseldorf, Germany
modaresi@cs.uni-duesseldorf.de

Alexander Askinadze
Institute of Computer Science
Heinrich Heine University
Düsseldorf
D-40225 Düsseldorf, Germany
askinadze@cs.uni-duesseldorf.de

Stefan Conrad
Institute of Computer Science
Heinrich Heine University
Düsseldorf
D-40225 Düsseldorf, Germany
conrad@cs.uni-duesseldorf.de

ABSTRACT

We developed an approach to automatically predict the personality traits of Java developers based on their source code for the PR-SOCO challenge 2016. The challenge provides a data set consisting of source code with their associated developers' personality traits (neuroticism, extraversion, openness, agreeableness, and conscientiousness). Our approach adapts features from the authorship identification domain and utilizes features that were specifically engineered for the PR-SOCO challenge. We experiment with two learning methods: linear regression and k-nearest neighbors regressor. The results are reported in terms of the Pearson product-moment correlation and root mean square error.

CCS Concepts

•Computing methodologies → Artificial intelligence; Natural language processing;

Keywords

Computational personality recognition; five factor model; Java source code

1. INTRODUCTION

Author profiling is a research field that deals with the prediction of user properties (e.g., age and gender prediction of an author [10]). The subfield computational personality recognition refers to an interdisciplinary field that incorporates computer science and psychology to automatically infer an author's personality based on his or her generated contents [4]. Although the generated contents can be of any form, we focus on textual contents in this work.

A popular personality model used in computational personality recognition is the *five factor model* [2]. According to this model, five fundamental traits exist that make up the human personality and each consists of several facets: neuroticism (anxiety, depression, angry hostility), extraversion (warmth, positive emotions, activity), openness (fantasy, aesthetics, values), agreeableness (trust, straightforwardness, compliance), and conscientiousness (competence, order, dutifulness).

Computational personality recognition has been applied to various domains, such as essays [8], tweets [7], and blogs [11]. An interesting but less studied application is the personality prediction of software developers based on their written source code. Unlike blogs and tweets, which are written (mostly) in natural languages, source code is written in a programming language that might not explicitly reveal the author's personality.

The study of software developers' source code has many practical applications. For instance, in the education sector for detecting plagiarism [1], in the law sector for cybercrime investigation [5], and in the technology sector to identify the expertise level of programmers [6]. To the best of our knowledge, there have been no studies on the automatic prediction of software developers' personalities based on their source code. Having a tool capable of predicting the personality of a software developer based on his or her open source projects (GitHub[1], Bitbucket[2], etc.) could dramatically improve the recruitment process of companies since software development requires teamwork and deciding if a programmer's personality fits the team is crucial for companies.

In this paper, we introduce a machine learning approach developed in the scope of the PR-SOCO [12] shared task to automatically identify the personality type of a Java developer based on his or her source code. Participants were provided with a training set consisting of Java sources codes of programmers annotated with the five previously discussed personality traits and with a test set. The aim of the PR-SOCO task is the development of approaches that predict the personality traits of programmers on the test set.

We investigated two classes of features: *structure* features dependent on the programming experience of the programmer (architecture design, code complexity, etc.) and *style*

[1]https://github.com/
[2]https://bitbucket.org/

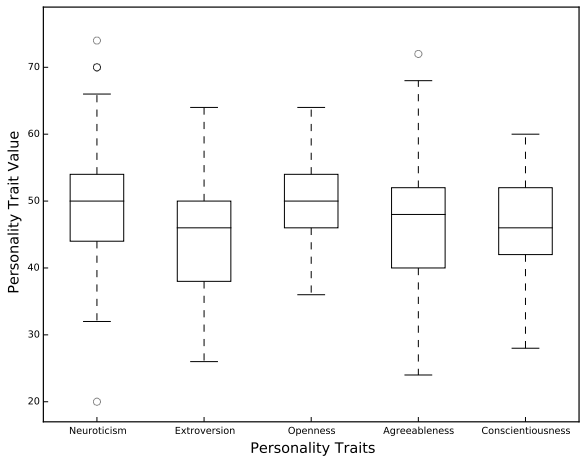

Figure 1: Distribution of personality traits in the training set

features related to the code layout that cannot be easily changed by IDEs (comment length, variable length, etc.). We intentionally ignored the *layout* features (line length, formatting style, etc.) as these features can be easily modified by IDEs using available formatting and code cleaning functionalities [3].

The remainder of the paper is structured as follows: Section 2 describes the PR-SOCO challenge and our contribution to solving it. The results of our approach are described in Section 3. We conclude and outline future work in Section 4.

2. APPROACH

In order to process the students' Java source code, we first created *knife*[3] which is an open-source wrapper for the two Java parsers *QDOX*[4] and *JavaParser*[5]. Knife parses source code into classes, methods, parameters, and variables and uses the Spark micro framework to provide the parsed code as JSON. Afterwards, *pisco*[6] consumes the parsed source code, extracts features, and uses machine learning to predict personality traits with linear regression and the k-nearest neighbors regressor.

2.1 Data

The data for the PR-SOCO challenge comprises solutions for different Java programming tasks that were uploaded by students and the results of their personality tests. Each of the five personality traits is represented by a value between 20 and 80. The students were allowed to upload more than one solution per programming task and to reuse code from previous exercises or from external resources. The training set comprises 49 data points and the test set contains 21 data points. It might be difficult to train classifiers and avoid outliers with such a low amount of data.

Figure 1 shows a boxplot for the personality traits in the training set. It can be observed that the median personality scores are between 46 and 50.

[3]https://github.com/pasmod/knife
[4]https://github.com/paul-hammant/qdox
[5]https://github.com/javaparser/javaparser
[6]https://github.com/Liebeck/pisco

The data was not cleaned by the organizers and, therefore, its quality varied. It sometimes contained debug output, empty classes, syntax errors or even Python code. Another influencing factor is that students occasionally used external code that was copied into the project, e.g., code from programming lectures at other universities. Since the focus of this challenge is the prediction of the students' personality types, a proper filtering step for external code seems reasonable. Otherwise, the prediction of the students' personality types can be influenced by other coder's personality types. Unfortunately, we were not able to perform a plagiarism check via web search.

2.2 Implemented Features

With the parsed source code from knife, we are able to implement several style and structure features for our machine learning approach.

2.2.1 Style Features

While naming conventions are certainly a controversial topic of debate for software developers (who each have their own programming style), we believe that the naming of classes, methods, fields and local variables is important for the understanding of the code. For instance, overly short or overly long variable names can be difficult to understand. Therefore, the length of such names might correlate with how thoughtful a developer was while writing source code. We decided to use the following style features:

F1: Length of method names

F2: Length of method parameter names

F3: Length of field names

F4: Length of local variables names in methods
An interesting observation is that the training data contains a solution from one student who used a local variable name that is 75 characters long while the mean length of local variable names for all students is 4.02 ($\sigma = 3.89$). Such an outlier can be problematic for linear regression.

2.2.2 Structure Features

We investigated ten structure features that we consider to be related to the developer's programming experience. A more experienced developer might tend to write shorter methods with fewer lines of code or less code in general.

F5: Number of classes

F6: Cyclomatic complexity
The cyclomatic complexity [9] is a software metric that calculates the number of linear independent paths in a program's control flow. We calculate the cyclomatic complexity per method by starting with an initial value of 1, which is increased for each occurence of control flow modifying keywords, such as *if* or *for*.

F7: Number of methods

F8: Number of method parameters

F9: Length of methods
We included the length of methods in our feature set

since long methods can be an indicator that the *single responsiblity principle* is violated and the methods could be refactored into multiple smaller methods. In our experiments, we tested the length of methods in terms of the number of lines and in terms of characters (without indentation).

F10: Number of fields per class

F11: Number of local variables in methods

F12: Duplicate code measure
We noticed that some students uploaded multiple solutions with very similar looking code. They copy pasted methods from one class to another while performing small changes to the code. This motivated us to check whether a student uploaded two methods that have a high overlap.[7]

The duplicate code measure was implemented as a binary feature. The code lines from all methods were tokenized and converted into bag-of-words models. Afterwards, we calculated the pairwise cosine similarity between all methods and considered two methods $m_i \neq m_j$ to be a duplicate of each other by comparing their similarity with a threshold τ:

$$\mathrm{DCM}(m_i, m_j) := \begin{cases} 1 & \text{if } \cos(m_i, m_j) > \tau \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

We empirically estimated $\tau = 0.9$. A student uploaded duplicate code if $\mathrm{DCM}(m_i, m_j) = 1$ for two of his or her methods $m_i \neq m_j$.

F13: Usage of IDE default template text
We noticed that some students did not remove or change default IDE text content and implemented this behavior as a binary feature.

F14: Ratio of external library usage
Developers are nowadays able to share libraries via dependency managers, which allow developers to use implementations of other developers without the need to write all the code from scratch. In the case of Java, code can be grouped into packages which can be imported. This feature calculates the ratio of imports from standard Java packages to all imports.

2.2.3 Miscellaneous Features

F15: Number of empty classes
We noticed that the submitted solutions sometimes contain empty classes. This might be an indicator of how thoroughly a programmer works or how important cleaning up source code is for him/her.

F16: Ratio of unparsable solutions
This feature captures that students uploaded code that is not valid Java code. A student's solution might contain syntax errors that made it unparsable for QDOX. This is especially the case where students uploaded debug output or Python code. This feature is implemented as the ratio of parsable to unparsable solutions. It reflects how careful the students were in following instructions or in testing if their code meets the specified requirements.

[7]This is not to be confused with a plagiarism check between the solutions of different students.

Although it might be useful to analyze code comments (e.g., the average comment length), we decided not to use features based on code comments since line and block comments may be polluted by code that was commented out.

2.3 Cross-Validation

Since most of our features are on a class or method basis, we need to aggregate their values to a vector representation of a fixed length in order to deal with different numbers of solutions, classes, fields, methods, and parameters. In order to make our features more robust against outliers, we first aggregate the values per solution with a summary statistic (e.g., mean, variance, range) and then calculate their mean. Given that the choice of a summary statistic is not apparent, we decided to choose it via cross-validation on the training set.

Additionally, we noticed different behaviors of the features depending on the personality trait. This encouraged us to estimate an optimal feature set for each personality trait individually. Since we have 16 features and the power set of all of these features contains too many combinations, it is not computationally feasible to search the entire feature space. First, we performed a cross-validation on the training set with all 16 features. Additionally, we experimented with subsets of our features and chose the subset that performed best during the 10-fold cross-validation on the training set.

3. EVALUATION

In total, 11 teams participated at the PR-SOCO shared task and submitted 48 runs.

3.1 Evaluation Metrics

Two evaluation metrics were proposed for the evaluation of the submissions. To measure the correlation between the predicted values and the gold standard values, the Pearson product-moment correlation coefficient (PC) was used. Moreover, the root mean square error (RMSE) was used to measure the average amount of prediction errors. For a vector $\boldsymbol{y} \in \mathbb{R}^n$ of truth values and its corresponding prediction vector $y \in \mathbb{R}^n$, the equations of the Pearson product-moment correlation and RMSE are shown in Equations 2 and 3 respectively:

$$r = \frac{\sum_{i=1}^{n}(\boldsymbol{y}_i - \bar{\boldsymbol{y}})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad (2)$$

where $\bar{\boldsymbol{y}}$ and $\bar{y}$ denote the average values of the vectors $\boldsymbol{y}$ and y respectively and n represents the number of data points.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\boldsymbol{y}_i - y_i)^2}{n}} \quad (3)$$

3.2 Results

To optimize the hyperparameters (meaning parameters that do not need to be learned as part of the model, e.g., summary statistics for features and parameters that have to be set manually for learning algorithms), we performed

an exhaustive 10-fold cross-validated grid search over all hyperparameters for each personality trait individually. We used k-nearest neighbors regressor (runs 3 and 4) and linear regression (runs 5 and 6), and optimized once to minimize RMSE (runs 4 and 5) and once to maximize the Pearson correlation (runs 3 and 6). After observing the results of the cross-validation, we noticed that none of the two learning algorithms could outperform the other one. As a result, we decided to choose the learning algorithm for each personality trait individually and chose the one with the higher cross-validation score on the training data. This resulted in two more runs since we once optimized for the Pearson correlation (run 1) and once for RMSE (run 2).

The task organizers also provided two baseline approaches: a bag of character 3-grams with frequency weight and an approach that always predicts the mean value observed in the training data [12].

The settings of the best runs, including the selected features and the applied learning algorithm, together with their corresponding RMSE values, are summarized in Table 1. Note that the numbers listed under *selected features* correspond to the feature indexes introduced in Section 2.2. It is observable that the k-nearest neighbors regressor has superior results over the linear regression method for all personality traits. As we discussed previously, several extracted features include outliers, which can cause large residual values by linear regression. By contrast, the k-nearest neighbors regressor is capable of coping with outliers and is preferred by the grid search.

It is also observable that the features length of field names (F3), duplicate code measure (F12), usage of IDE default template text (F13), and number of empty classes (F15) are among the most powerful predictors of personality traits.

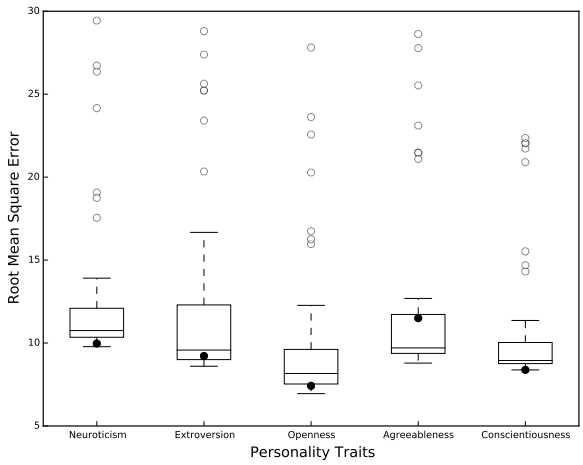

Figure 2: Root Mean Square Error Results

In Figure 2, we compare our results regarding the RMSE measure to the other participants. The results not included between the whiskers are considered as outliers and are represented by empty circles. For each personality trait, the filled circle indicates the RMSE values of our best runs. For all personality traits except *agreeableness*, our proposed approach had RMSE values lower than the median. In particular, we achieved the lowest RMSE among all participating teams for the personality trait *conscientiousness*.

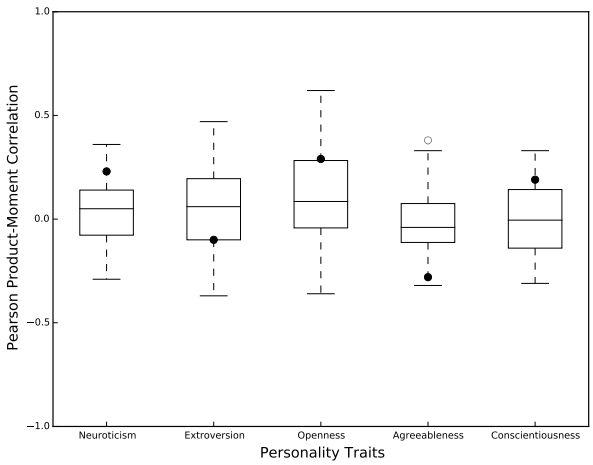

Figure 3: Pearson's Correlation Results

For comparison, we also provide the settings of the best runs regarding the Pearson correlation in Table 2. Similar to the case of RMSE, the features F3, F12, F13, and F15 were identified to result in higher Pearson correlations. For the personality traits *extroversion* and *agreeableness*, based on the grid search results, linear regression resulted in higher Pearson correlations in comparison to the k-nearest neighbors regressor. Nevertheless, linear regression results in negative correlation coefficients for both traits. The Pearson correlations of our best runs for the individual traits can be compared to the other submissions in Figure 3.

4. CONCLUSIONS

We presented our approach to automatically predict personality types in the five factor model from Java source code for the PR-SOCO challenge 2016. Our architecture consists of the two components knife and pisco which we made publicly available on GitHub. We used knife to parse the source code and pisco to extract features and to predict personality traits.

We achieved the best root mean squared error for the personality trait conscientiousness among all 11 participating teams. For the personality traits neuroticism and openness, our best runs ranked 3rd and 9th, respectively, based on 48 runs. Our RMSE result for the trait extroversion was better than the median. Unfortunately, the results in the dimension openness were not satisfactory. The results in terms of the Pearson correlation were mixed since we achieved positive and negative correlations.

In our future work, we want to crawl external resources in order to determine if pieces of the source code are plagiarized. We also want to evaluate non-linear machine learning approaches. During our data analysis, we identified that the developers sometimes used more than one natural language, for instance in comments or in variable names. We would like to investigate this behavior for possible correlations with personality types. In our work, we ignored layout features since they can easily be modified by an IDE. However, we could investigate if the developer is consistent in using the auto formatter of his or her IDE.

Table 1: Selected features for the best runs according to RMSE

Personality Trait	Selected Features																Method	RMSE
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
Neuroticism	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	k-NN	9.97
Extroversion												✓			✓		k-NN	9.22
Openness			✓									✓	✓		✓		k-NN	7.42
Agreeableness	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	k-NN	11.5
Conscientiousness			✓									✓	✓		✓		k-NN	8.38

Table 2: Selected features for the best runs according to the Pearson correlation

Personality Trait	Selected Features																Method	PC
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
Neuroticism	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	k-NN	0.23
Extroversion	✓	✓			✓			✓		✓	✓	✓	✓		✓	✓	LR	-0.05
Openness			✓									✓	✓		✓		k-NN	0.29
Agreeableness	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	LR	-0.28
Conscientiousness			✓									✓	✓		✓		k-NN	0.19

5. ACKNOWLEDGMENTS

This work was partially funded by the PhD program *Online Participation*, supported by the North Rhine-Westphalian funding scheme *Fortschrittskollegs*„ by the German Federal Ministry of Economics and Technology under the ZIM program (Grant No. KF2846504), and by the IST-Hochschule University of Applied Sciences. Computational support and infrastructure were provided by the "Centre for Information and Media Technology" (ZIM) at the University of Düsseldorf (Germany).

6. REFERENCES

[1] A. Ahtiainen, S. Surakka, and M. Rahikainen. Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises. In *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, pages 141–142. ACM, 2006.

[2] P. T. Costa and R. R. McCrae. *The NEO personality inventory manual*. Psychological Assessment Ressources, 1985.

[3] H. Ding. Extraction of Java Program Fingerprints for Software Authorship Identification. Master's thesis, Faculty of the Graduate College of the Oklahoma State University, 2002.

[4] G. Farnadi, G. Sitaraman, S. Sushmita, F. Celli, M. Kosinski, D. Stillwell, S. Davalos, M.-F. Moens, and M. De Cock. Computational personality recognition in social media. *User Modeling and User-Adapted Interaction*, 26(2):109–142, 2016.

[5] G. Frantzeskou and S. Gritzalis. Source Code Authorship Analysis for Supporting the Cybercrime Investigation Process. In *ICETE 2004, 1st International Conference on E-Business and Telecommunication Networks*, pages 85–92, 2004.

[6] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill. A Degree-of-Knowledge Model to Capture Source Code Familiarity. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 385–394. ACM, 2010.

[7] J. Golbeck, C. Robles, M. Edmondson, and K. Turner. Predicting Personality from Twitter. In *SocialCom/PASSAT*, pages 149–156. IEEE, 2011.

[8] F. Mairesse, M. A. Walker, M. R. Mehl, and R. K. Moore. Using Linguistic Cues for the Automatic Recognition of Personality in Conversation and Text. *J. Artif. Int. Res.*, 30(1):457–500, Nov. 2007.

[9] T. J. McCabe. A Complexity Measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976.

[10] P. Modaresi, M. Liebeck, and S. Conrad. Exploring the Effects of Cross-Genre Machine Learning for Author Profiling in PAN 2016. In *Working Notes of CLEF 2016 - Conference and Labs of the Evaluation forum*, pages 970–977, 2016.

[11] J. Oberlander and S. Nowson. Whose thumb is it anyway? Classifying author personality from weblog text. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, COLING-ACL '06, pages 627–634. Association for Computational Linguistics, 2006.

[12] 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.