CAPS-PRC: A System for Personality Recognition in Programming Code

Notebook for PAN at FIRE16

Ivan Bilan
Center for Information and Language Processing
Ludwig Maximilian University of Munich
Oettingenstr. 67
Munich, Germany
ivan.bilan@gmx.de

Eduard Saller
Center for Information and Language Processing
Ludwig Maximilian University of Munich
Oettingenstr. 67
Munich, Germany
eduard@saller.io

Benjamin Roth
Center for Information and Language Processing
Ludwig Maximilian University of Munich
Oettingenstr. 67
Munich, Germany
beroth@cis.uni-muenchen.de

Mariia Krytchak
Department of Psychology
Ludwig Maximilian University of Munich
Leopoldstr. 13
Munich, Germany
mariia.krytchak@gmx.de

ABSTRACT

This paper describes the participation of the CAPS-PRC system developed at the LMU Munich in the personality recognition shared task (PR-SOCO) organized by PAN at the FIRE16 Conference. The machine learning system uses the output of a Java code analyzer to investigate the structure of a given program, its length, its average variable length and also it takes into account the comments a given programmer wrote. The comments are analyzed by language independent stylometric features, including TF-IDF distribution, average word length, type/token ration and more. The system was evaluated using Root Mean Squared Error (RMSE) and Pearson Product-Moment Correlation (PC). The best run exhibited the following results: Neuroticism (RMSE - 10.42, PC - 0.04), Extroversion (RMSE - 8.96, PC - 0.16), Openness (RMSE - 7.54, PC - 0.1), Agreeableness (RMSE - 9.16, PC - 0.04), Conscientiousness (RMSE - 8.61, PC - 0.07).

Keywords

machine learning; Big Five personality traits; source code analysis; abstract syntax tree

1. INTRODUCTION

The main purpose of the task is to investigate whether it is possible to predict personality traits of programmers based on the source code written by them [8]. Previous research has identified the relationship between personality factors and computer programming styles having used different measures of personality [2] [4]. The task considers the Big Five personality traits which were assessed by the NEO-PI-R Inventory [5] to form the training set [8]: extroversion, emotional stability/neuroticism, agreeableness, conscientiousness, and openness to experience. The Big Five Model, i.e. five broad fairly independent dimensions, encompasses all personality traits and is considered to describe the personality in a comprehensive way. The NEO-PI-R Inventory is a statistically reliable and valid tool that operationalizes the Big Five Model through self/other-assessment and is set in various cross professional and cross cultural contexts to describe the personality.

2. EXPERIMENTAL SETUP

2.1 Approaching the problem

Based on the available research on the Big Five psychological traits [5] [3], we can see that the traits are considered to be independent of each other. For this reason, each psychological trait was viewed and analyzed individually. Figures 1 to 5 show the distribution of the training set for each psychological trait by author. Table 1 shows the mean trait distribution.

Since each programmer/author has submitted more than one program, we approach the problem from two different angles:

1) the feature vectors are extracted for each programmer, by first extracting them for each program and then averaging all the underlying feature vectors into one single feature vector for the author. The classifier learns based on a single feature vector for each author, where the author represents one sample in the dataset.

2) the classifier is trained at the level of programs. Each program inherits the trait value of its author. The feature vectors are extracted for each program and then the classifier regards each program as a training instance. To get back to

Figure 1: Author Distribution, Agreeableness

Figure 2: Author Distribution, Conscientiousness

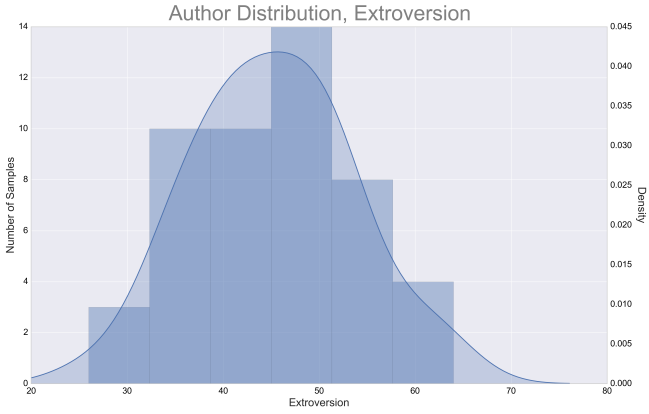

Figure 3: Author Distribution, Extroversion

Figure 4: Author Distribution, Neuroticism

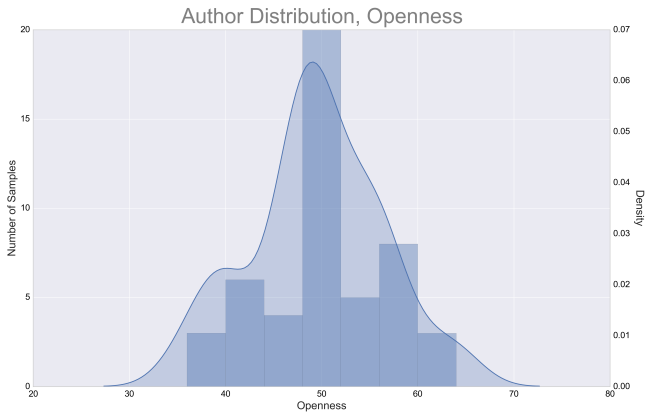

Figure 5: Author Distribution, Openness

the level of authors (while the final prediction should be done for the author), the predictions are averaged for each program belonging to a certain author. The final result is a single prediction for each author based on the predictions produced for each underlying program.

2.2 Feature Extraction

2.2.1 Abstract syntax tree

We use a grammar γ specifically designed for the analysis of a programming language β, which in the context of the task was the Java programming language. The grammar γ combined with a parser ρ provides a semantic representation of the source code called an abstract syntax tree (AST). Compared to normal parse trees there are some potential advantages. First, the generation of an AST can be interpreted as a normalization step of our feature generation. In contrast to the original source code, which has inconsistencies like whitespace characters or other unneeded characters, the AST represents a concise version of a given program. This also makes the generation of meta-features (compositions of different base features) more simple, due to the strict representation of all, to the compiler important, parts of the program. Additionally, the representing syn-

Trait	Mean Value	Standard Deviation
Agreeableness	47.02	8.95
Conscientiousness	46.37	6.46
Extroversion	45.22	8.19
Neuroticism	49.92	11.15
Openness	49.51	6.68

Table 1: Mean Trait Distribution, Training Set

Distribution / Dataset	Train Set	Test Set
Min. Programs per Author	6	14
Mean Programs per Author	37	37
Max. Programs per Author	122	109
Total Number of Programs	1790	772
Total Number of Authors	49	22

Table 2: Programs per Author

tax tree is not necessarily bound by the original syntactic rules of the original programming language β which allows for generalizations of the source code to occur.

In our approach, we use the frequency distribution of all known entities in the grammar to build a feature list for a given program. This shallow use of the AST provides 237 features for a given source code analysis. Some examples would be the *Type* of variables or the nature of a statement(do, for, while, etc.) The implementation of the AST is made possible with the help of ANTLR parser [6].

2.2.2 Custom Features

In addition to the AST, we used additional features for the source code and also the comments. The following is an exhaustive list of all additional features used.

Code-based features: length of the whole program (in lines of code, in characters), the average length of variable names, what indentation the programmer is using (tabs or spaces).

Comment-based features: type/token ratio, usage of punctuation marks, TF-IDF, the frequency of comments (block comments and inline comments separately), average word length.

Author-level based features: number of programs submitted (see Table 2), average length of programs in lines of code.

2.3 Classification

We experimented with a number of Regression classifiers like Linear Regression, Ridge Regression, Logistic Regression and Gradient Boosted Regression. In addition, we have tried to detect the outliers with the RANdom SAmple Consensus (RANSAC). The final system implementation did not use RANSAC, since it delivered worse results. Although, this technique should be further investigated with a bigger dataset.

We have submitted our final runs based on two machine learning algorithms: Gradient Boosted Regression and Multinomial Logistic Regression. Furthermore, Gradient Boosted Regression was evaluated on the level of authors and the programs level, while the Multinomial Logistic Regression was implemented on the level of authors.

Personality Traits	Author-based	
	Evaluation Metric	
	RMSE	PC
Agreeableness	9.17	-0.12
Conscientiousness	8.83	-0.31
Extroversion	9.55	-0.1
Neuroticism	10.28	0.14
Openness	7.25	-0.1

Table 3: Results of the Multinomial Logistic Regression Approach

The first classification approach is based on Gradient Boosted Regression with least squares regression as its loss function, 1100 estimators, 5 as the maximum depth of the individual regression estimators, and the learning rate of 0.1. This approach also utilized χ^2 test for the feature selection to choose only the best 200 features from the AST feature extraction pipeline. This approach was implemented using the scikit-learn Python library [7].

The second approach is based on the Multinomial Logistic Regression model with the l2-regularized squared loss as its objective function. That is, each feature was multiplied with a trait-specific weight, and the result of this linear combination was the input to a sigmoid activation. As the output of this prediction is in the range [0,1], we re-scaled the trait values in the training data to the same range for computing the squared loss.

Training was done using stochastic gradient descent with constant learning rate, and parameters were tuned on the held-out development set using random search. The search space of the parameters was: learning rate $\in 0.01, 0.1, 1$, number of training epochs $\in \{10, 20, 50, 100, 200, 500\}$, regularization $\in \{0, 0.001, 0.01, 0.1, 1\}$, (mini-)batch size $\in \{1, all\}$. The best configuration was: learning rate: 1, training-epochs: 2, regularization: 0.6, batch-size: all. This approach was developed with theano Python library [1].

3. EXPERIMENTAL RESULTS

The dataset included 49 programmers in the training set (with 1790 programs in total) and 22 programmers in the test set (772 programs). Final evaluation was done with two different evaluation metrics: Root Mean Squared Error (RMSE) and Pearson Product-Moment Correlation (PC). In the Gradient Boosted Regression approach (GBR Approach), the system was tuned to maximize both of these metrics at the same time, while the Multinomial Logistic Regression one (MLR Approach) concentrated on RMSE. Table 3 gives a detailed overview of the results achieved using Multinomial Logistic Regression at the level of authors. Table 4 shows the results achieved using the Gradient Boosted Regression approach at the level of authors and the level of programs.

In general, the results are low using both RMSE and PC and only slightly outperform the performance of the baseline approaches (see Table 5). Two baselines have been provided by the task organizers [8]:

1) 3-gram character representation.

2) always predict the mean trait value of the training dataset.

Personality Traits	Author-based		Program-based	
	Evaluation Metric			
	RMSE	PC	RMSE	PC
Agreeableness	10.89	-0.05	9.16	0.04
Conscientiousness	8.9	0.16	8.61	0.07
Extroversion	11.18	-0.35	8.96	0.16
Neuroticism	12.06	-0.04	10.42	0.04
Openness	7.5	0.35	7.54	0.1

Table 4: Results of the Gradient Boosted Regression Approach

Personality Traits	3-gram characters		Mean value	
	Evaluation Metric			
	RMSE	PC	RMSE	PC
Agreeableness	9.00	0.20	9.04	0.00
Conscientiousness	8.47	0.17	8.54	0.00
Extroversion	9.06	0.12	9.06	0.00
Neuroticism	10.29	0.06	10.26	0.00
Openness	7.74	-0.17	7.57	0.00

Table 5: Baseline Approaches

4. CONCLUSIONS

This paper describes the system that given a source code collection of a programmer, identifies their personality traits. While the RMSE and PC scores proved promising during development, further investigation suggested the dataset may be too small to create an effective machine learning system. The compiler style feature generation process using ASTs combined with several custom features could serve as future baselines for similar tasks.

4.1 Future Work

The task would benefit greatly from an expanded training corpus (more samples per programmer, more programmers). The value distribution of the training set is also an important point. The current training set exhibits normal distributed scores for each Big Five trait. A more robust system could be created when using an equal number of samples within low, mid and high value range.

Additionally, further feature engineering, additional statistical analysis of the AST output, and transferring strategies of other NLP tasks involving syntax trees onto the current task could improve the system.

5. REFERENCES

[1] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[2] C. Bishop-Clark. Cognitive style, personality, and computer programming. *Computers in Human Behavior*, 11(2):241–260, 1995.

[3] O. P. John and S. Srivastava. The big five trait taxonomy: History, measurement, and theoretical perspectives. *Handbook of personality: Theory and research*, 2(1999):102–138, 1999.

[4] Z. Karimi, A. Baraani-Dastjerdi, N. Ghasem-Aghaee, and S. Wagner. Links between the personalities, styles and performance in computer programming. *Journal of Systems and Software*, 111:228–241, 2016.

[5] F. Ostendorf and A. Angleitner. *Neo-PI-R: Neo-Persönlichkeitsinventar nach Costa und McCrae*. Hogrefe, 2004.

[6] T. Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, Nov. 2011.

[8] 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.