Shallow recurrent neural network for personality recognition in source code

Yerai Doval
Grupo COLE, Departamento de Computación
E.S. de Enxeñaría Informática, Universidade de Vigo
Campus As Lagoas, 32004 – Ourense (Spain)
yerai.doval@uvigo.es

Carlos Gómez-Rodríguez
Grupo LYS, Departamento de Computación
Facultade de Informática, Universidade da Coruña
Campus de Elviña, 15071 – A Coruña (Spain)
cgomezr@udc.es

Jesús Vilares
Grupo LYS, Departamento de Computación
Facultade de Informática, Universidade da Coruña
Campus de Elviña, 15071 – A Coruña (Spain)
jvilares@udc.es

ABSTRACT

Personality recognition in source code constitutes a novel task in the field of author profiling on written text. In this paper we describe our proposal for the PR-SOCO shared task in FIRE 2016, which is based on a shallow recurrent LSTM neural network that tries to predict five personality traits of the author given a source code fragment. Our preliminary results show that it should be possible to tackle the problem at hand with our approach but also that there is still room for improvement through more complex network architectures and training processes.

CCS Concepts

•Applied computing → Law, social and behavioral sciences; Psychology; Document analysis; •Human-centered computing → Text input; •Social and professional topics → User characteristics; •Computing methodologies → Natural language processing; Neural networks;

Keywords

personality recognition, source code, recurrent neural network, LSTM

1. INTRODUCTION

Written text can tell us a lot about its author. Demographic information such as age, gender or specific personality traits of the author can be inferred by a human expert by the sole observation of a written text fragment [7]. This task is called *author profiling*, and it can be also applied to other channels such as speech or body language. But detecting the patterns which allow for this kind of information extraction is not restricted to humans, as we will see in this work.

Source code is another form of written text, and it is becoming very accessible as software developers are now able to easily publish their work on the Web through services such as Github[1] or Bitbucket.[2] Although more constrained and formal than natural language, source code text may also have something to tell us about its author, as there is still room for personal preferences in its writing. For instance, some coders tend to use block delimiters even when they are not necessary, or to add a certain number of blank lines in order to clearly separate two function declarations. Morevoer, variable and function names are custom made by the coder, and commentaries include information in natural language. Therefore, it sounds reasonable to take advantage of this type of patterns to attempt to extract information about the author of a source code fragment, which constitutes a novel task in the author profiling field.

In this work, we describe our contribution to the Personality Recognition in SOurce COde (PR-SOCO) shared task [13], held in conjunction with the FIRE 2016. The objective of this task is quantifying five personality traits about the author of a given source code fragment, namely, the standard traits from the Big Five Theory [6]: extroversion, emotional stability/neuroticism, agreeableness, conscientiouness and openness to experience. To achieve this, we propose using a shallow recurrent neural network that, taking as input the sequence of bytes in an input source code text, will try to predict the five values for the corresponding traits of its author. By reading the most elementary unit available for encoded text, the byte (in most cases directly aligned with individual characters), we seek to find all possible useful patterns carved deep into the text. Furthermore, with this approach we are not limiting our models to those patterns a human can grasp, but we are enabling the neural network to extract any information it may consider useful for the task.

The results obtained with our shallow networks are encouraging with respect to the root mean squared error metric (RMSE), which is aligned with the smoothed mean absolute error criterion employed in our training process. However, they do not perform so well for Pearson Correlation (PC), which we have not considered at this time. We have also found that the use of more layers in our networks can improve their performance, agreeing with previous work [17].

2. RELATED WORK

There has been a recent surge of interest on author profiling related to personality recognition [14, 3].

For written text, traditional author profiling approaches tend to rely on lexical and syntactical features, such as identification of key words, part-of-speech tags [1] or n-grams [10] paired with statistical models such as Hidden Markov Models. There is also work which studies the application of

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

these traditional techniques on short informal texts, which often translates into lower performance figures than those obtained for regular texts [18].

However, author profiling is not restricted to written text. Mairesse et al [12] extend this type of analysis to speech, where features such as sound frequencies and the duration of pauses made by the speaker are considered. Biel et al [2] go one step further by analysing Youtube videos and adding what they call "nonverbal cues" to the feature set, which take into consideration the different types of motion that can be observed in the video. There are even approaches that analyse the structure and topology of the social network of subjects [16].

Regarding the psychological aspects of this work, the proposed task relies on the so-called Big Five Theory [6] to establish the personality traits to be predicted: extroversion, neuroticism, agreeableness, conscientiousness, and openness to experience. It is worth noting that although in trait theory there are more than five traits, the most extended theoretical approaches reduce its total number to five, as in the case of the Big Five Theory, or even to just three: neuroticism, extraversion and psychoticism [15].

3. THE PROPOSED APPROACH

Written text, either natural language or source code, can be viewed as a sequence of basic elements such as sentences, words or characters, to name a few possibilities. Given a particular domain, we can choose the sequential view of the input text which best fits our needs. In our case, source code is full of reserved keywords such as `if`, `return` or `while`, so a word-based approach may seem appropriate at first, as the word vocabulary seems to be relatively fixed and reduced. However, the problem then comes with the custom names given by the coder to classes, variables, functions, etc. which have an unpredictable nature and do not fit well in a strict vocabulary approach. Furthermore, it would be interesting that the vocabulary of sequence elements was as small as possible since this affects the required size of the input layer of our models. In order to keep things simple, we will not follow a word-level or character-level approach but a pure byte-level approach, thus limiting the size of the vocabulary to 256 possible byte values.

To process these byte sequences we will use recurrent neural networks, as they are a perfect fit for sequential data [5]. Thus, each byte from the input sequence is fed to the network at each time step through the input layer, which transforms byte values into internal representations that can be manipulated by the hidden layers of the network. Moreover, the output of these hidden layers is not only influenced by the current input but also by some of the information retained from those bytes processed at previous time steps. This is achieved thanks to the recurrent connections added to the neurons in these layers of the network. Once the final byte from the input sequence has been processed, the output from the last hidden layer corresponding to the last time step is then used to perform a linear transformation and produce as a result a vector of five values, each of them corresponding to a particular personality trait (as described in Section 2). In order to achieve this, the network had to be accurately trained to return relevant values at its output and not just random garbage. In this case, we have configured it to minimize the difference between its obtained output and the desired one for each input sequence. More precisely, we have used a smoothed mean absolute error as the training criterion of the network, which uses a squared term if the absolute element-wise error falls below one, making it less sensitive to outlier data and preventing exploding gradients, a common problem in neural network training [8].

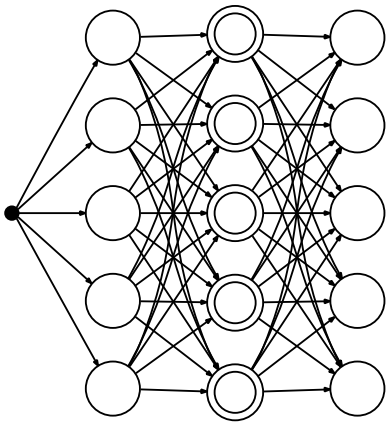

Figure 1: Simplified view of the neural network used. The first layer should have 256 neurons, one per possible input byte value. The second layer, the only hidden layer in this case, is conformed by recurrent LSTM units. The last layer is the output layer, and has exactly 5 neurons, one for each trait we want to calculate.

It is worth noting that, in contrast with previous work (see Section 2), our approach does not require a feature engineering phase as neural networks, in their training processes, reflect the most interesting features from the input domain in the values of their parameters, sometimes referred to as weights.

Lastly, we opted for feeding the network with input instances (sequences) which are independent from each other, so that the important dependence relationships (patterns) between elements (bytes) of a sequence may be observed by our model. For this reason, we have constructed sequences from whole source code packages. As these sequences can be quite long (see Section 4), traditional recurrent networks may have problems to recall important information extracted at the beginning of the input sequence while they are processing the last elements of it. In order to address this limitation, we use long short term memory (LSTM) units as the neurons in the hidden layers of the network [9]. These units pack a memory cell and other elements that manipulate its contents, thus enabling them to remember important information from the distant past of an input sequence. See Figure 1 for a simplified visual representation of this model.

4. EXPERIMENTS

Our models were implemented using the scientific framework Torch [4] and the recurrent neural network library `torch-rnn` [11]. We took advantage of the GPU computing capabilities of these resources using an Nvidia GTX Titan X. For further implementation details, the source code will be made available at https://cloud.wyffy.com/index.php/s/EphokbtRuQ43BWc.

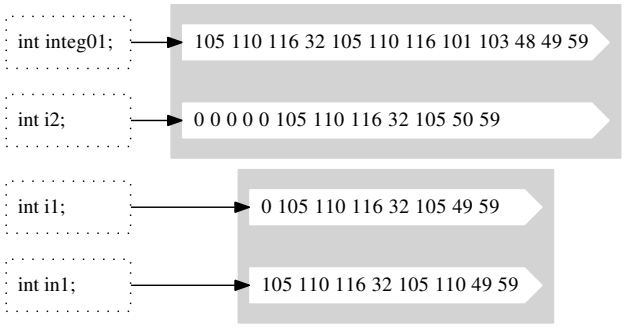

Figure 2: Text is represented as sequences of byte values which are then gathered into batches (grey filled rectangles) where they are padded with zeros at the beginning.

Training and validation

First of all, we have preprocessed the training corpus given by the PR-SOCO organization to best fit the training procedure of the neural network. In this vein, we have merged the personality information (i.e. the personality trait values) from each one of the 49 developers right after the end marker of the package of their source code files. This way our input format contains the input sequence to the network above the marker and the desired output one line below. Then we merged all resulting files into a single one and shuffled the instances, resulting in a total of 1600 instances. The validation dataset was built taking the first 141 instances from the resulting file, leaving the rest for training.

The shortest meaningful sequence in the training corpus has length 34, the longest 27654 and the average one is approximately 4823 characters long. Regarding the personality scores in the training corpus, their values fall in the range 20–80 and their means are: 49.92 for neuroticism, 45.22 for extroversion, 49.51 for openness, 47.02 for agreeableness and 46.37 for conscientiousness.

In order to benefit from the processing power of the GPU, we gathered input sequences into batches. Since all sequences in a given batch must have the same length, we padded the shorter sequences with zeros at the beginning. Unfortunately, this was not exactly the case throughout our experiments, and until very recently the padding was being added to the end of shorter sequences instead, giving rise to a bug were these sequences were automatically discarded in the training process. This bug did not affect experiment settings with a batch size of 1.

We show in Figure 2 how some sample input texts are represented as sequences of byte values which are then gathered into batches where they are appropriately padded with zeros at the beginning.

The training process consists of 100 full cycles (epochs) through the training corpus. The time needed to accomplish this depends on the complexity of the network and the batch size used. As an example, one epoch in a network with two hidden layers of 300 neurons each and a batch size of 1 can take up to 4.8 hours while using a batch size of 10 reduces the training time to 2.6 hours. Similarly, adjusting the batch size to 10, a network formed by a 300 neurons hidden layer needs 2 hours to train through one epoch. It is important

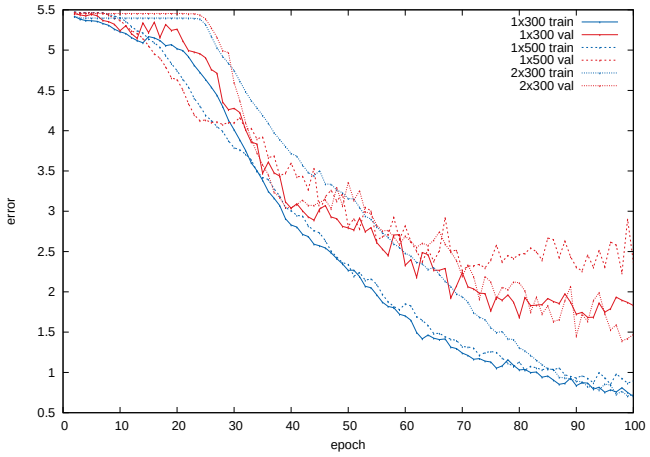

Figure 3: Training and validation error evolution through 100 training epochs for three different model configurations.

	#hlayers	batch size
run01-v2	1	10
run02	2	10
run03-v2	1	20
run04	1	1
run05-v2	1	1

Table 1: Number of hidden layers and batch size of the models used for the test runs. All of them have 300 neurons per hidden layer. The difference between *run04* and *run05-v2* is the training time, greater in the latter case.

to note that this figures would be lower if we did not run multiple training processes in parallel.

In Figure 3 we show our preliminary experiments to attest for the capacity of our models to tackle the task at hand. Although the observed behaviour in training time of these models was acceptable and invited us to use them against the test corpus (which we will describe shortly), they were affected by the padding bug mentioned earlier and cannot be considered as clear evidence of the performance of the models. Since the batch size was established to 10, the bug caused the models to train with a tenth of the total training and validation instances. In any case, as we can see in the graph, it seems beneficial for the generalization capabilities of a neural network trained for this task to add at least one extra hidden layer to its architecture (steady training data fit and lower final validation error), while adding neurons to a sole hidden layer results in a counterproductive measure.

Testing and official results

The test corpus supplied by the PR-SOCO organization did not undergo a preprocessing stage such as the one described above. In this case we have to evaluate 21 developers whose source code is fragmented in 750 test instances. The maximum sequence length observed is 33550, the minimum 114 and the mean 3743.

For the five runs performed on the test data, we have used five different models differing in the number of hidden layers and batch size to be employed, which are related in Table 1.

	N	E	O	A	C
run01-v2	11.99	**11.18**	12.27	**10.31**	**8.85**
run02	12.63	11.81	**8.19**	12.69	9.91
run03-v2	**10.37**	12.5	9.25	11.66	8.89
run04	29.44	28.8	27.81	25.53	14.69
run05-v2	11.34	11.71	10.93	10.52	10.78
task mean	12.75	12.27	10.49	12.07	10.74

Table 2: Official PR-SOCO RMSE results over 5 runs. Personality traits: (N)euroticism, (E)xtroversion, (O)pennes, (A)greeableness and (C)onscientiousness. *run02* is the only run affected by the batch padding bug.

	N	E	O	A	C
run01-v2	-0.01	0.09	-0.05	0.2	0.02
run02	-0.18	0.21	-0.02	-0.01	-0.3
run03-v2	**0.14**	0.0	0.11	-0.14	0.15
run04	-0.24	**0.47**	-0.14	**0.38**	**0.32**
run05-v2	0.05	0.19	**0.12**	-0.07	-0.12
task mean	0.04	0.06	0.09	-0.01	-0.01

Table 3: Official PR-SOCO PC results over 5 runs. Personality traits: (N)euroticism, (E)xtroversion, (O)pennes, (A)greeableness and (C)onscientiousness.

All of them have 300 neurons per hidden layer and have been trained with the whole training corpus, including the validation part. Note that the difference between *run04* and *run05-v2* is the training time, longer in the latter case. The only run affected by the padding bug was *run02*.

In Tables 2 and 3 we can see our official results obtained for the PR-SOCO task. In general, the correlation scores are quite low while the RMSE figures are acceptable (considering that they beat the task average) except for *run04*, whose better results in correlation might be attributed to a mere coincidence. On the other hand, we see that RMSE scores for *run02* are quite good despite being the only case affected by the batch padding bug mentioned above. This fact seems to be related with the benefits provided by the extra network layer that the corresponding model has with respect to the rest. We can also observe, in the difference between *run04* and *run05-v2*, that allowing the model to train for longer periods of time is indeed useful to attain good performance. Finally, at this time the data available do not allow us to extract any particular conclusion about the influence of the batch size on our results.

It is worth noting that, unfortunately, we could not re-run the 2-hidden layer network without the padding bug against the test corpus because of time constraints. Nevertheless, in order to confirm the hypothesis that adding an extra layer to the network is beneficial to its performance, we have conducted some *a posteriori* experiments with the training corpus. In Figure 4 we can see how the 2-hidden layer network obtains, once again, better generalization capabilities than the 1-hidden layer network.

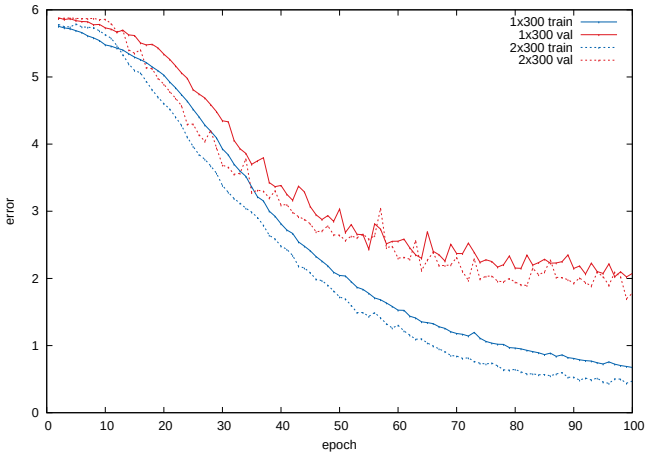

Figure 4: Training and validation error evolution for 1 and 2-hidden layer networks not affected by the padding bug.

5. CONCLUSIONS

Source code is a form of written text which has been becoming very accessible in recent years. While more constrained and formal than natural language due to its very nature, it also allows for some personal preferences to pour down into its structure and content, giving rise to the possibility of author profiling on it.

In this paper we have shown our proposal for personality recognition in source code. Viewing such text as a sequence of characters (or bytes), we have used shallow recurrent neural networks as our personality trait predictors. In order to maximize the pattern detection capabilities of our model, we have fed entire source code packages as sequence inputs to the network. The network learning criterion was a smoothed mean absolute error, less sensitive to outliers than RMSE or the mean absolute error.

Given the encouraging results obtained, we think that our approach may be a viable one to tackle this problem. On one hand, the RMSE figures obtained, which are aligned with the criterion we were optimizing for, are positive considering that we have used a shallow network, whose expressivity power is limited, with large input sequences. On the other hand, we have found some hints pointing at a better performance in the case of using deeper neural networks and training them for longer periods of time, which may constitute immediate ways of improving our results.

As future lines of work, we will try to improve our results by adding more layers to our neural network—in a one-by-one fashion until we see no more significant improvement—, and also by introducing a new training criterion that considers the correlation between instances. Another interesting research line would be the study and visualization of the activation mechanisms which occur within the network at evaluation time in order to try to interpret the patterns, or features, that the model has previously extracted during the training phase. In other words, to analyse the behaviour of the network to try to observe human interpretable patterns and thus distil the knowledge condensed in the network.

6. ACKNOWLEDGMENTS

This work has been partially funded by the Spanish Ministerio de Economía y Competitividad through projects FFI2014-51978-C2-1-R and FFI2014-51978-C2-2-R, and by Xunta de Galicia through an Oportunius program grant.

We gratefully acknowledge NVIDIA Corporation for the donation of a GTX Titan X GPU used for this research.

7. REFERENCES

[1] S. Argamon, M. Koppel, J. Fine, and A. R. Shimoni. Gender, genre, and writing style in formal written texts. *TEXT*, 23(3):321–346, 2003.

[2] J.-I. Biel, O. Aran, and D. Gatica-Perez. You Are Known by How You Vlog: Personality Impressions and Nonverbal Behavior in Youtube. In *ICWSM*, 2011.

[3] F. Celli, B. Lepri, J.-I. Biel, D. Gatica-Perez, G. Riccardi, and F. Pianesi. The Workshop on Computational Personality Recognition 2014. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 1245–1246. ACM, 2014.

[4] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[5] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994.

[6] P. T. Costa and R. R. MacCrae. *Revised NEO personality inventory (NEO PI-R) and NEO five-factor inventory (NEO FFI): Professional manual*. Psychological Assessment Resources, 1992.

[7] D. P. Crowne. *Personality theory*. Don Mills, Ont.: Oxford University Press, 2007.

[8] R. Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] J. Houvardas and E. Stamatatos. N-gram feature selection for authorship identification. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 77–86. Springer, 2006.

[11] N. Léonard, S. Waghmare, and Y. Wang. RNN: Recurrent library for torch. *arXiv preprint arXiv:1511.07889*, 2015.

[12] 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. *Journal of Artificial Intelligence Research*, 30:457–500, 2007.

[13] 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.

[14] F. Rangel, P. Rosso, M. Potthast, B. Stein, and W. Daelemans. Overview of the 3rd Author Profiling Task at PAN 2015. In *CLEF*, 2015.

[15] E. H. E. SBG. Manual of the Eysenck personality questionnaire, 1975.

[16] J. Staiano, B. Lepri, N. Aharony, F. Pianesi, N. Sebe, and A. Pentland. Friends don't lie: inferring personality traits from social network structure. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 321–330. ACM, 2012.

[17] M. Telgarsky. Benefits of depth in neural networks. *CoRR*, abs/1602.04485, 2016.

[18] C. Zhang and P. Zhang. Predicting gender from blog posts. Technical report, Technical Report. University of Massachusetts Amherst, USA, 2010.