UMUTeam at AI-SOCO'2020: Source Code Authorship Identification based on Character N-Grams and Author's Traits

José Antonio García-Díaz[a], Rafael Valencia-García[a]

[a]*Facultad de Informática, Universidad de Murcia, Campus de Espinardo, 30100 Murcia, Spain*

Abstract

The purpose of authorship attribution is to determine who is the author of a certain work by comparing features of the said work with other documents for which the authorship is known. In the software forensics domain, revealing who is the author behind a piece of software can prevent intellectual property infringements as well as prevent and detect cheating in academic courses. In this paper, we describe the participation of the UMUTeam from the University of Murcia in the PAN's shared task: Authorship Identification of SOurce COde (AI-SOCO'2020). Our proposal is grounded on a mixture of statistical features and heuristics that we called author's traits. From the three runs submitted, we achieve a best accuracy of 91.16% over the testing dataset reaching the sixth place in the official ranking. It is worth noting that our proposal, which relies in traditional machine-learning classifiers, outperforms competitive baselines based on state of the art deep-learning transformers such as RoBERTa.

Keywords

Authorship Identification, Supervised Machine-learning, Feature Engineering

1. Introduction

In a broad sense, authorship identification deals with the problem of identifying who is the author behind certain unidentified work by comparing and finding similarities between that work with other works whose authorship is known. Authorship identification has many applications. For example, it can be used to provide evidences that the person listed as the author of certain work was not actually the person who wrote it, unveiling cases of ghostwriting, fraud or plagiarism. Moreover, authorship identification can help to unmask who is the author of anonymous threatening messages. Applied to the manufacturing software domain, authorship identification (1) reduces the chances of committing intentional or unintentional plagiarism, and (2) can used for tracking malicious software [1].

In academia, e-learning platforms are easing for both students and teachers to do their work at home, which also promotes self-discipline and self-evaluation skills during software learning [2]. Moreover, as technology evolves quickly, these skills are also needed to employees in technological companies in order to adapt themselves to the industry and an increasingly competitive market. In this context, online judges and contest platform systems allow the

Forum for Information Retrieval Evaluation, December 16-20, 2020, Hyderabad, India
EMAIL: joseantonio.garcia8@um.es (J. A. García-Díaz); valencia@um.es (R. Valencia-García)
ORCID: 0000-0002-3651-2660 (J. A. García-Díaz); 0000-0003-2457-1791 (R. Valencia-García)

participants to compile, execute, and evaluate their approaches unsupervised [3]. Nevertheless, the feeling of not being under close surveillance in a classroom or in the office can encourage people to cheat [4] and, therefore, online judges and contest platform systems should incorporate mechanisms to detect evidences of cheating in order to ease the work of the reviewers.

With the aim of improving current authorship identification techniques, the Forum for Information Retrieval Evaluation (FIRE'2020) proposed *Authorship Identification of SOurce COde (AI-SOCO)*, a PAN shared task focused on "*uncovering the author who wrote some piece of code*" [5]. This task consists of, given a pre-defined set of source codes and authors, infer who the author is for each source code. In this paper we describe our contribution with the submissions of three runs based on a machine-learning models trained with statistical features based on character n-grams and some heuristics that we called author's traits.

The rest of the paper is organised in the following way. First, in Section 2 different approaches for solving authorship identification are discussed in detail. Then, the reader can find a description of the materials and methods employed in Section 3. Next, the results achieved by our runs compared with the baselines proposed by the organisers and the runs of the rest of the participants are shown and discussed in Section 4. Finally, the conclusions and further work are presented in Section 5.

2. Background information

The first approaches regarding authorship identification relied in (1) attribute counting-metric systems, that included metrics to measure the number code lines, unique operands, or variables declared; and (2) structure metrics, in which abstract representations of the program structure are compared [6]. Nowadays, the majority of works found in the bibliography employed machine-learning approaches. For example, in [7] the authors developed an authorship identification system that extracts both statistical features such as word n-grams as well as some hand-craft features regarding the code structure. The authors found that some of the hand-crafted features reflected "*explicit and implicit personal programming preference patterns of and between keywords, identifiers, operators, statements, methods and classes*". In [8], Bander et al. employed Recurrent Neuronal Networks (RNNs) based on traditional and bidirectional Long-Short Term Memory networks (BiLSTM) from the Abstract Syntax Tree (AST) without the need of hand-crafted features. Another work concerning RNNs is described at [9], in which the authors employed a Gated Recurrent Units (GRU) evaluated on two datasets achieving, respectively, an accuracy of 69.1% and 89.2%. Another approach can be found at [10], in which the authors compare Latent Semantic Analysis (LSA) with re-use detection models to measure cross-language similarity in source code. In [11], the contribution of the authors is two fold. On the one hand, they present two language-agnostic models that outperformed language-specific systems on datasets of three popular programming languages. On the other, they identify some weakness in source-code datasets for authors profiling, highlighting the fact that the environment in which the programmers write code (which they refer as *work context*) influences their style by forcing some decisions such as naming conventions. In addition, they argue that (1) existing datasets do not consider fair code collaboration, and (2) the fact that the author style could vary among time. Fair code collaboration issue is addressed at [12], in which the authors proposed working

on authorship of source code segments. Their approach involved a stacking ensemble method composed of deep neural networks and machine learning classifiers achieving promising results. An interesting approach to address authorship identification can be found at [13], in which the authors reversed the problem and proposed a black-box attack against authorship attribution of source code by performing semantics-preserving code transformations in order to create variations of the source code that tricks machine-learning solutions in order to induce false attributions. The idea behind this approach is to create source code that can be used in adversarial learning.

Author profiling task regarding the software domain has also caught the attention in scientific workshops and conferences. In 2014, the shared task *Detection of SOurce COde Re-use* [14] was proposed as a PAN shared task, which consisted in the identification of source code re-use from an unbalanced dataset that consisted in code written in C and Java. A total of five teams participated in this task with a total of 17 runs. Another shared task is described at [15], in which the participants were required to predict the author's personality from five big traits from source codes written in Java. At the end of the task, a total of 48 runs from 11 participants were sent.

3. Materials and methods

In this work, we describe the three runs submitted by the UMUTeam to the AISOCO'2020 shared task. Accordingly, in this section we describe briefly the dataset provided by the organisers of the task (see Section 3.1), and we describe our proposal pipeline (see Section 3.2).

3.1. Dataset description

The organisers of the AI-SOCO'2020 shared task compiled the dataset from Codeforces[1]. This is a popular website that hosts programming contests in which the participants can propose solutions to solve the proposed challenges. Each time the participants send their programs, the online judge labels each response as accepted or attach some labels to indicate an incorrect response as well as that some constraints regarding time and memory were not fulfilled.

The dataset provided to the participants consisted in 100,000 accepted source codes written in C++ from a total of 1000 users. The dataset is balanced so there are no authors underrepresented, which eases the classification task. The whole corpus was released into three datasets: (1) training (with 50,000 source codes), (2) development (with 25,000 source codes), and (3) testing (with 25,000 source codes).

3.2. Pipeline

In a nutshell, our pipeline can be described as follows: First, each source code is automatically inspected in order to create two alternative versions: one that contains only the comments and another with the source code without the comments. Second, we extract the features which involves character n-grams and the author's traits. Next, we apply feature selection techniques

[1] https://codeforces.com/

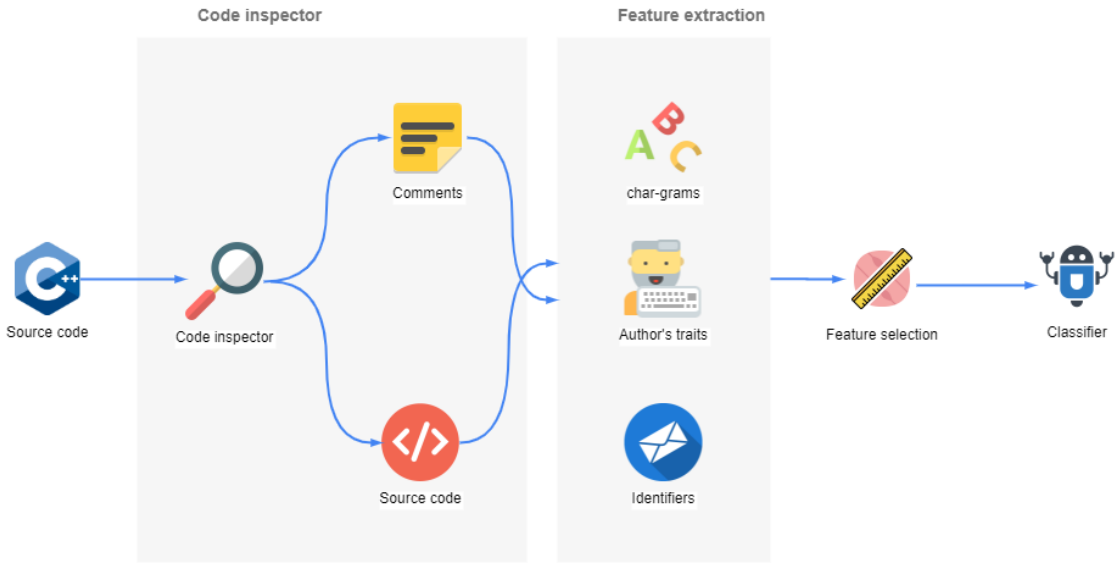

Figure 1: Pipeline of the UMUTeam proposal at AISOCO'2020

in order to discard highly correlated features and, finally, we train a machine-learning classifier with the training dataset that we used to predict the labels of the development and testing datasets. This pipeline is depicted in Figure 1.

3.2.1. Feature extraction

Regarding the feature extraction, our contribution is grounded in the combination of character n-grams and author's traits features. On the one hand, we use character n-grams as the foundation of our proposal as they can capture stylistic patterns and nuances at lexical, syntactical, and structural level [16]. Specifically, we extracted combinations between 1 and 8 length applying TF-IDF with Sub-linear TF scaling. We preserve the letter case both of the source code and the comments and we set an upper-bound of 400 characters n-grams per length. That is, we select the best character n-grams of length 1, the best character n-grams of length 2, and so on until character n-grams of length 8. This resulted in 3200 character n-grams. On the other hand, after a manual analysis of some instances of the source code provided, we manually handcraft features to capture author's traits. These features are organised in the following categories: (1) stylometrics, which measure, for example, the average length of code blocks, comments, or words in uppercase; (2) code traits, that reflect how the authors focused the problem. In this sense, we measure the number of nested loops or the use of keywords in the code; (3) non-ASCII characters, to measure languages other than English, such as Arabic, Indian, or Russian; and (4) ASCII-ART, characters employed in the creation of decorative forms that users employed as signature. In addition, since some source codes included contact data such as nicknames, URLs, or Twitter accounts, we compiled a Bag of Words composed of the tokens that followed the words *name* and *author* in the block comments as well as certain topics such as mentions to blessings. A list of this features can be found in Table 1.

Table 1
Author trait features and categories

category	feature	description
stylometry	line_length_average	Average length of line in source code, comments and both
	code_length	Total length of source code, comments and both.
	uppercase_letters	Percentage of uppercase letters in the source code, comments and both
	comment_block_average_length	Average length of a code block
	tokens_of_specific_length	nº of tokens of 2-11 length
	assignations	Number of assignations with spaces " = ", or together "="
	vars_with_underscores	Number of variables with underscores "_"
code	nested_loops	Total of nested loops of depth 2, 3 and 4
	block_length_average	Block length average between brackets, parenthesis and square brackets
	string_length_average	Length average of the strings variables
	keywords_in_code	Number of keywords in source code
non-ascii	non_ascii_percentage	Percentage of non-ascii characters
	language_spanish	Percentage of Spanish characters
	language_asian	Percentage of Asian characters
	language_russian	Percentage of Russian characters
	language_arabic	Percentage of Arabic characters
ascii-art	decorative_symbols	Tokens surrounded by decorative characters
	asciiart_blocks	ASCII-art with blocks
	asciiart_delimiters	ASCII-art with delimiters
	asciiart_braille	ASCII-art with braille
topics	religious	Mentions to gods or blessings

It is worth noting that some of these features were previously employed by our research group in UMUTextStats tool to conduct different text classification task such as (1) Sentiment Analysis [17, 18], (2) satire identification [19], or (3) misogyny detection [20] with good results.

3.2.2. Feature selection and machine-learning classifier

Once all the features were compiled, they were filtered by applying a feature selection consisting of removing those features that have the same value in all samples in order to discard similar character n-grams. Then, we evaluate different supervised machine-learning classifiers, such as Random Forest, Multi Layer Perceptron, K-Neighbours, Support Vector Machines with linear kernels, and multinomial Naive Bayes. Out of these, the best accuracy was achieved with Random-Forest classifier. This algorithm was built using SciKit in Python and the hyper-parameters were: (1) 800 number of trees in the forest, (2) a maximum depth of each tree of 100, (3) with bootstraping disabled (so the whole dataset is used to build each random tree), (4) considering $sqrt(n\_features)$ features for the best split, (5) with min_samples_leaf equal to 2,

Table 2
Results of the UMUTeam runs with the three runs with the Development and Testing datasets

Run	Accuracy Development	Accuracy Testing
character n-grams + author-traits	0.9140	0.9116
character n-grams	0.9130	0.9112
author-traits	0.5628	0.5647

and (6) the default min_samples_split of 2 (as default).

4. Results

At the beginning of the task, the organisers provided three baselines. The first baseline consisted in predicting a random author with an expected accuracy of 0.1%. The second baseline consisted in count English printable characters and to apply a logistic regression model, which achieved an accuracy of 29.252%. The last baseline consisted in a K-nearest neighbours with the TF-IDF of 10,000 features. This baseline achieved an accuracy of 62.128%. After showing the final results of the competition, the organisers included two more baselines based on RoBERTa [21], in which they vary the complexity of the neural networks layers.

The evaluation of the proposal was performed using the Accuracy metric (see Equation 1) as it can defined as the relationship between items correctly identified: True Positives (TP) and True Negatives (TN) with the total of instances classified including False Positives (FP) and False Negatives (FP).

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{1}$$

First, we show the results of our submissions comparing the accuracy of the development and testing datasets in Table 2. As it can be observed, the best accuracy with the development dataset was 91.40% and an accuracy of 91.16% for the testing dataset. In both cases, the best run is based on the combination of the statistical features and the author's traits. However, we can observe that a similar accuracy can be achieved on both approaches by using only the character n-grams model (run-2). When looking the results of the author traits separately (run 3), we achieved an accuracy of 56.28% which beats the second baseline proposed by AI-SOCO, consisted in a Logistic Regression model on the printable characters (29.92%) as it is not far from the TF-IDF K-nearest neighbours baseline (62.78%).

In Figure 4, we show the Information Gain (IG) of some features regarding the author's traits. We decided to include features for all the categories in order to see if they contribute equality or not regarding authorship identification. In view of the results, we can conclude that those metrics regarding with stylometry are the most discriminating features. For example, it seems that the average length of the source code, comments, and the percentage of uppercase letters have more influence in authorship identification than in determining what source codes resolve the same problem. A similar guess can be done when looking the author's traits from the *code* category. We can observe that the number of nested loops is relevant regarding the author profiling. This fact can be related to the way in which programmers focus on how to resolve

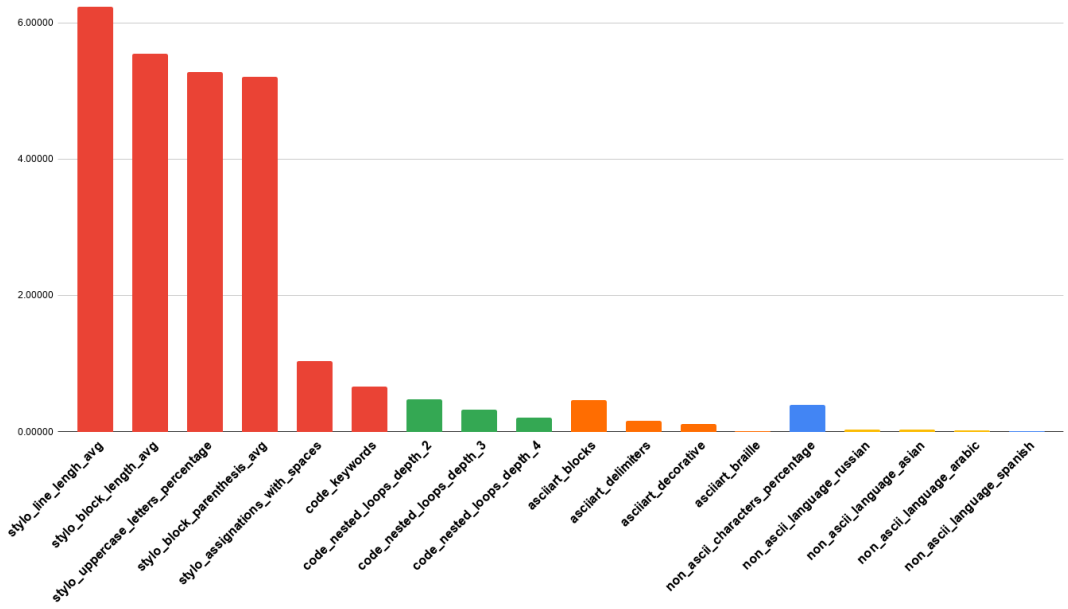

Figure 2: Information gain of the author's traits

a problem efficiently in order to win the contest. Finally, we can observe that IG decreases regarding the usage of non-ASCII characters in order to forecast the country of origin of the programmer, or the usage of ASCII-art as signature in their comments.

The comparison of our proposal with the rest of the baselines and the rest of the participants are shown in Table 3. It is worth noting that this table contains only the official runs. Some extra runs were sent after the official end-date, highlighting the results achieved by the Twist Bytes team achieving an accuracy of 94.4%.

As it can be observed from our results compared with the official participants and the baselines (see Table 3), our proposal reaches the sixth place and outperforms baselines based on deep-learning transformers such as RoBERTa. We achieve an accuracy of 91.16% which indicates that our proposal is quite reliable regarding the identification of the authors. Moreover, our proposal outperforms by far the first baselines provided and the first baseline of the two generated with RoBERTa. However, the second baseline based on RoBERTa, with six layers, outperforms our proposal with an 92.88% of accuracy.

5. Conclusions and Further work

In this paper we describe the participation of the UMUTeam at the AISOCO'2020 shared task for author identification of source code. This was our first participation on one task of this kind and we are very happy with the results. It draws our attention that traditional machine-learning classifiers could outperform some models based on the state-of-the-art transformers, which

Table 3
Official results of the AISOCO'2020 shared task. The first column represents the ranking position. An asterisk indicate that this result is a baseline

#	Team	Accuracy
1	AlexCrosby	0.9511
2	yang1094	0.9428
3	mutaz	0.9336
*	AI-SOCO RoBERTa Code Baseline (6L, 12H)	0.9288
4	zz	0.9219
5	FSU_HLJIT	0.9157
6	**UMUTeam**	**0.9116**
*	AI-SOCO RoBERTa Code Baseline (1L, 96H)	0.9102
7	Abdalrhman	0.9088
8	bits_nlp_2020	0.9064
9	gaojiaming	0.8616
10	chanchal	0.8295
11	panyawut-sr	0.8208
12	aupatsara-wa	0.8202
13	meghana	0.8120
14	TaeyongSeong	0.8091
15	christopher-w	0.7452
*	AI-SOCO TFIDF KNN Baseline	0.6278
*	AI-SOCO Characters Logistic Baseline	0.2992
*	AI-SOCO Random Baseline	0.0008
*	Twist Bytes	0.9440
*	ken_tt	0.8067

suggests that the feature extraction and selection plays a crucial role in authorship attribution, as our proposal achieves competitive results reaching the sixth position among the official results with an accuracy of 91.16%.

The average accuracy of the participants (without the baselines) is 87.05% with a standard deviation of 0.0623, which indicates that all the participants achieved very competitive results in a difficult task regarding author identification. We consider, however, that the results of this task would be lower if the source code was provided to us without comments. Moreover, as the dataset was compiled from online sources, we suppose that the dataset does not contain any kind of cheating that could make it hard this task. We think that it is not common that participants in online contents do not know each other and that prevents direct plagiarism between the authors as could happen in academic courses.

During our review of literature we found that not all research take comments into consideration. Comments make code easier for understand and they are a valuable data source for author profiling as a comment may include copyright info, explanations about implementation decisions, or even arbitrary patterns that act as section delimiters [22]. However, they are not always available as, for example, in low-level languages which are common in malware software. As source comments can include text written in natural language, it is feasible to extract linguistic features from comments in order to create a fingerprint of the author's writing style. For example, with the detection of slang, jokes, or the usage of figurative language [23],

it is possible to deduce which is the cultural and social background of the author. As further work, we will improve the author traits for authorship identification as we consider that these features provide more interpretability of the results that character n-grams. In addition, we will continue investigating on author identification. In this sense, we will explore programming collaborative environments such as GitHub or Stack Overflow.

Acknowledgments

This work has been supported by the Spanish National Research Agency (AEI) and the European Regional Development Fund (FEDER/ERDF) through projects KBS4FIA (TIN2016-76323-R) and LaTe4PSP (PID2019-107652RB-I00). In addition, José Antonio García-Díaz has been supported by Banco Santander and University of Murcia through the Doctorado industrial programme.

References

[1] E. H. Spafford, S. A. Weeber, Software forensics: Can we track code to its authors?, Computers & Security 12 (1993) 585 – 595. URL: http://www.sciencedirect.com/science/article/pii/016740489390055A. doi:https://doi.org/10.1016/0167-4048(93)90055-A.

[2] J. Gravill, D. Compeau, Self-regulated learning strategies and software training, Information & Management 45 (2008) 288 – 296. URL: http://www.sciencedirect.com/science/article/pii/S0378720608000475. doi:https://doi.org/10.1016/j.im.2008.03.001.

[3] S. Wasik, M. Antczak, J. Badura, A. Laskowski, T. Sternal, A survey on online judge systems and their applications, ACM Computing Surveys (CSUR) 51 (2018) 1–34.

[4] G. Watson, J. Sottile, Cheating in the digital age: Do students cheat more in online courses?, Online Journal of Distance Learning Administration 13 (2010).

[5] A. Fadel, H. Musleh, I. Tuffaha, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, P. Rosso, Overview of the PAN@FIRE 2020 task on Authorship Identification of SOurce COde (AI-SOCO), in: Proceedings of The 12th meeting of the Forum for Information Retrieval Evaluation (FIRE 2020), 2020.

[6] K. L. Verco, M. J. Wise, Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems, in: ACM International Conference Proceeding Series, volume 1, 1996, pp. 81–88.

[7] C. Zhang, S. Wang, J. Wu, Z. Niu, Authorship identification of source codes, in: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, Springer, 2017, pp. 282–296.

[8] B. Alsulami, E. Dauber, R. Harang, S. Mancoridis, R. Greenstadt, Source code authorship attribution using long short-term memory based networks, in: S. N. Foley, D. Gollmann, E. Snekkenes (Eds.), Computer Security – ESORICS 2017, Springer International Publishing, Cham, 2017, pp. 65–82.

[9] C. Qian, T. He, R. Zhang, Deep learning based authorship identification, Department of Electrical Engineering, Stanford, CA (2017).

[10] E. Flores, A. Barrón-Cedeño, L. Moreno, P. Rosso, Cross-language source code re-use detection using latent semantic analysis., J. UCS 21 (2015) 1708–1725.

[11] E. Bogomolov, V. Kovalenko, A. Bacchelli, T. Bryksin, Authorship attribution of source code: A language-agnostic approach and applicability in software engineering, 2020. arXiv:2001.11593.

[12] P. Mahbub, N. Z. Oishie, S. M. Rafizul Haque, Authorship identification of source code segments written by multiple authors using stacking ensemble method, in: 2019 22nd International Conference on Computer and Information Technology (ICCIT), 2019, pp. 1–6.

[13] E. Quiring, A. Maier, K. Rieck, Misleading authorship attribution of source code using adversarial learning, in: 28th {USENIX} Security Symposium ({USENIX} Security 19), 2019, pp. 479–496.

[14] E. Flores, P. Rosso, L. Moreno, E. Villatoro-Tello, On the detection of source code re-use, in: Proceedings of the Forum for Information Retrieval Evaluation, 2014, pp. 21–30.

[15] F. Rangel, F. González, F. Restrepo, M. Montes, P. Rosso, Pan@ fire: overview of the pr-soco track on personality recognition in source code, in: Forum for Information Retrieval Evaluation, Springer, 2016, pp. 1–19.

[16] J. Houvardas, E. Stamatatos, N-gram feature selection for authorship identification, in: International conference on artificial intelligence: Methodology, systems, and applications, Springer, 2006, pp. 77–86.

[17] J. A. García-Díaz, M. Cánovas-García, R. Valencia-García, Ontology-driven aspect-based sentiment analysis classification: An infodemiological case study regarding infectious diseases in latin america, Future Generation Computer Systems 112 (2020) 614–657. doi:https://doi.org/10.1016/j.future.2020.06.019.

[18] J. M. Ruiz-Martínez, R. Valencia-García, F. García-Sánchez, et al., Semantic-based sentiment analysis in financial news, in: Proceedings of the 1st International Workshop on Finance and Economics on the Semantic Web, 2012, pp. 38–51.

[19] M. del Pilar Salas-Zárate, M. A. Paredes-Valverde, M. Á. Rodriguez-García, R. Valencia-García, G. Alor-Hernández, Automatic detection of satire in twitter: A psycholinguistic-based approach, Knowledge-Based Systems 128 (2017) 20–33.

[20] J. A. García-Díaz, M. Cánovas-García, R. Colomo-Palacios, R. Valencia-García, Detecting misogyny in spanish tweets. an approach based on linguistics features and word embeddings, Future Generation Computer Systems 114 (2020) 506–518.

[21] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, arXiv preprint arXiv:1907.11692 (2019).

[22] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, in: 2013 21st international conference on program comprehension (icpc), Ieee, 2013, pp. 83–92.

[23] M. del Pilar Salas-Zárate, G. Alor-Hernández, J. L. Sánchez-Cervantes, M. A. Paredes-Valverde, J. L. García-Alcaraz, R. Valencia-García, Review of english literature on figurative language applied to social networks, Knowl. Inf. Syst. 62 (2020) 2105–2137. URL: https://doi.org/10.1007/s10115-019-01425-3. doi:10.1007/s10115-019-01425-3.