SoftId: An autoencoder-based one-class classification model for software authorship identification

Mihaiela Lupea, Anamaria Briciu, Istvan Gergely Czibula, Gabriela Czibula

Department of Computer Science, Babeș-Bolyai University Cluj-Napoca, Romania

Outline

Outline

Authorship attribution

Definition: Authorship attribution (AA) is the task of determining the most likely author of a given text

Formalization of the AA problem:

- closed-set configuration (predefined number of authors/classes)
- open-set configuration (open test space to unknown authors/classes)

Software authorship attribution

Identify the author of a code fragment.

Problem relevance
Software authorship identification applications in software development: *software quality*, *legacy software systems*, *software archaeology*, *fraud/plagiarism detection activities in education*

In **software engineering**:

- practical use in multiple scenarios
- e.g. maximize the benefit of the code review process given time and other constraints by using AA model to select or prioritize code to review

Proposed model: *SoftId*

Autoencoder-based model to solve the software authorship attribution problem (SAA) in an open-set configuration.

Dataset: three subsets from Google Code Jam data set (3, 5 and 12 "original" developers) and additional "unknown" instances

Representation: TF-IDF, LSI

Contributions

1. development of an autoencoder-based one-class classification model (*SoftId*) that solves the SAA problem in an open-set configuration
2. representation of source code using natural language processing techniques
3. performance improvement over other one-class classifiers like *OSVM*

Research questions

RQ1 How to design an autoencoder-based one-class classifier for solving software authorship identification as an open-set-recognition problem?

RQ2 What is the relevance of the textual representation of source codes in discriminating between original (known) and other (unknown) software developers?

RQ3 Which of the two corpus-based representations, *term frequency - inverse document frequency)* (TF-IDF) or *Latent Semantic Indexing* (LSI), is better suited for our approach?

Outline

Related work

- **Software authorship identification**: popular domain with various approaches [KKG+20];
- **textual representations of source codes**: [ARA+19] *(document embeddings used to identify the author of a program)*, [SAS14] *(character N-gram based LSA model to create low approximation of data & obtain document pair similarities)*, [MM00] *(LSA to identify similarities between pieces of source code to assist in program understanding)*, [BVE15] *(TF-IDF and LSA compared in task of detecting semantic re-implementations)*
- **open-set configuration**: [BTGD21] *(data set: Victorian literature)*, [KS04] *(AA formalized as true one-class classification problem)*

Autoencoders (AE)

- deep learning models used in medical data analysis, image analysis, bioinformatics and other fields
- self-supervised learning technique

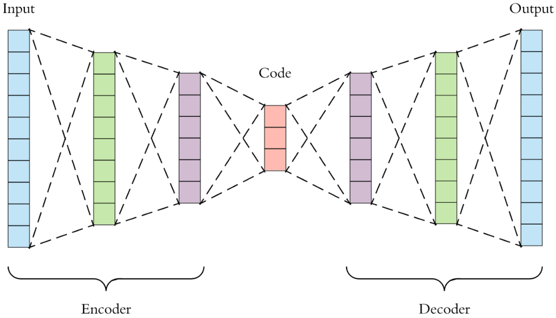

Figure: Autoencoder (AE) model[1]

[1] https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Outline

Problem statement

Formalization as an **open-set binary supervised classification** problem.

- set of k known software developers (authors) $\mathcal{S}d = \{Sd_1, Sd_2, \dots Sd_k\}$
- $\mathcal{SC} = Sc_1 \cup Sc_2 \cup \dots \cup Sc_k$ a set of software codes, known to be written by the given k authors
- **GOAL**: approximate a function $t : \mathcal{SC} \rightarrow \{``original", ``other"\}$ that maps a software code $sc \in \mathcal{SC}$ to either the "original" class (formed by the known developers from $\mathcal{S}d$) or the "other" class

The *SoftId* model

- the *SoftId* classifier consists of an autoencoder trained to encode patterns from the software codes belonging to authors from the set $\mathcal{S}d$
- at testing time, classifier will be able to decide if a given source code is authored by a *known* (developer from $\mathcal{S}d$) or an *unknown* one

Data preprocessing & representation

Data preprocessing

Tokenization

Data representation

- TF-IDF (*term frequency-inverse document frequency*)
- LSI (*Latent Semantic Indexing*)

Training (I)

Autoencoder A trained on $\mathcal{S}c = \bigcup_{i=1}^{k} Sc_i$ (the software codes authored by all the developers from $\mathcal{S}d$)

Train-validation-test split

- 70% will be used for *training*
- 20% will be used for *validation*
- 10% will be used for *testing*

Loss function

$L(\tilde{x}, x) = \frac{1}{m} \sum_{j=1}^{m} (\tilde{x}_j - x_j)^2$

x represents the m-dimensional input

$\tilde{x}$ represents the model's m-dimensional output

Training (II)

AE architectures

- for `TF-IDF` vectors of size 4000: input_layer + 2048-1024-512-256-512-1024-2048
- for `LSI` vectors of size 300: input_layer + 256-128-64-32-64-128-256

Model details

- hidden layers use ReLU activation function
- encoding layer uses linear activation
- network trained using **stochastic gradient descent** + Adam optimizer
- mini-batch perspective
- early stopping criterion - loss convergence on validation set is monitored (min_delta = 0.000025)

Testing & evaluation: Classification

Algorithm Classification for the testing source code sc.

function CLASSIFY($\mathcal{S}d, A, sc$)

Require:

$\mathcal{S}d$ - the set of original software developers; A - the AE trained to recognize the developers from $\mathcal{S}d$;

sc - the testing instance (source code) to be classified

Ensure:

return the predicted class ("original" or "other")

$vec_{sc} \leftarrow$ the vector representation of sc

$p_{other}(sc) = 0.5 + \frac{D(vec_{sc}, \widehat{vec_{sc}}) - \tau}{2 \cdot (D(vec_{sc}, \widehat{vec_{sc}}) + \tau)}$ /* Compute the probability that sc belongs to the "other" class*/

if $p_{other}(sc) \geq 0.5$ **then**

$c \leftarrow$ "other"

else

$c \leftarrow$ "original"

end if

return c

end function

Testing & evaluation: Evaluation

- cross-validation methodology: train/validation/test split repeated 10 times
- within each split, selection of “other” instances also repeated 10 times
- performance metrics:
  1. accuracy *(Acc)*
  2. precision *(Prec)* for the “original” class
  3. *Recall*
  4. *F*1-score
  5. specificity *(Spec)*
  6. Area under the ROC curve (*AUC*) [Faw06]
  7. *Area under the Precision-Recall curve* (*AUPRC*)

Outline

Dataset description

Subsets of the Google Code Jam [Goo] (GCJ) data set are used.

- subset of 3 original developers (709 software programs)
- subset of 5 original developers (1110 software programs)
- subset of 12 original developers (2325 software programs)

- software programs belonging to the "other" class randomly selected from the remaining instances in GCJ

Results (I)

Table: Performance metrics obtained by evaluating *SoftId* classifier on the Google Code Jam data set. 95% CI are used for the results.

No. of original authors	N-grams	TF-IDF representation							LSI representation						
		Acc	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*	*Acc*	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*
3	**5-grams**	0.947 ±0.006	1.000 ± 0.000	0.941 ± 0.007	0.970 ± 0.003	1.000 ± 0.000	**0.971** ± 0.003	0.971 ± 0.003	0.932 ±0.012	1.000 ±0.000	0.926 ±0.013	0.961 ±0.007	1.000 ±0.000	0.963 ±0.007	0.963 ±0.07
	6-grams	0.943 ±0.010	1.000 ± 0.000	0.937 ± 0.011	0.967 ± 0.006	1.000 ± 0.000	0.969 ± 0.006	0.969 ± 0.006	0.936 ±0.013	1.000 ±0.000	0.930 ±0.015	0.964 ±0.008	1.000 ±0.000	**0.965** ±0.007	0.965 ±0.007
	8-grams	0.939 ±0.014	1.000 ± 0.000	0.933 ± 0.016	0.965 ± 0.009	1.000 ± 0.000	0.966 ± 0.008	0.966 ± 0.008	0.936 ±0.016	1.000 ±0.000	0.930 ±0.017	0.964 ±0.010	1.000 ±0.000	**0.965** ±0.009	0.965 ±0.009
	10-grams	0.926 ±0.015	1.000 ± 0.000	0.919 ± 0.017	0.957 ± 0.009	1.000 ± 0.000	0.959 ± 0.008	0.959 ± 0.008	0.923 ±0.013	1.000 ±0.000	0.916 ±0.015	0.956 ±0.008	1.000 ±0.000	0.958 ±0.007	0.958 ±0.007
5	**5-grams**	0.943 ±0.013	0.995 ±0.002	0.943 ±0.014	0.968 ±0.007	0.950 ±0.016	0.946 ±0.015	0.969 ±0.008	0.929 ±0.015	0.999 ±0.001	0.923 ±0.016	0.959 ±0.009	0.988 ±0.007	0.955 ±0.012	0.961 ±0.008
	6-grams	0.941 ±0.017	0.994 ±0.001	0.940 ±0.018	0.966 ±0.010	0.947 ±0.010	0.944 ±0.014	0.967 ±0.010	0.933 ±0.020	0.998 ±0.001	0.928 ±0.022	0.962 ±0.012	0.982 ±0.008	0.955 ±0.015	0.963 ±0.011
	8-grams	0.947 ±0.013	0.996 ±0.001	0.945 ±0.014	0.970 ±0.007	0.962 ±0.009	**0.954** ±0.012	0.971 ±0.007	0.934 ±0.016	0.999 ±0.001	0.928 ±0.018	0.962 ±0.010	0.988 ±0.005	**0.958** ±0.012	0.963 ±0.009
	10-grams	0.937 ±0.013	0.996 ±0.001	0.935 ±0.015	0.964 ±0.008	0.964 ±0.010	0.949 ±0.012	0.965 ±0.008	0.921 ±0.014	0.998 ±0.001	0.915 ±0.015	0.954 ±0.008	0.983 ±0.005	0.949 ±0.010	0.956 ±0.008
12	**5-grams**	0.915 ±0.007	0.965 ±0.003	0.941 0.008	0.953 ±0.004	0.656 ±0.027	0.798 ±0.017	0.953 ±0.005	0.902 ±0.008	0.979 ±0.002	0.912 ±0.009	0.944 ±0.005	0.798 ±0.016	0.855 ±0.013	0.945 ±0.005
	6-grams	0.914 ±0.008	0.971 ±0.002	0.934 ±0.009	0.952 ±0.005	0.716 ±0.021	0.825 ±0.015	0.952 ±0.005	0.903 ±0.010	0.982 ±0.002	0.910 ±0.011	0.945 ±0.006	0.834 ±0.017	**0.872** ±0.014	0.946 ±0.006
	8-grams	0.904 ±0.012	0.976 ±0.002	0.917 ±0.013	0.945 ±0.007	0.772 ±0.020	0.845 ±0.017	0.946 ±0.008	0.912 ±0.009	0.981 ± 0.002	0.920 ±0.009	0.950 ±0.006	0.823 ±0.023	0.871 ±0.016	0.951 ±0.006
	10-grams	0.873 ±0.011	0.982 ±0.002	0.877 ±0.012	0.926 ±0.007	0.838 ±0.016	**0.857** ±0.014	0.929 ±0.007	0.874 ±0.011	0.980 ±0.003	0.880 ±0.011	0.927 ±0.007	0.820 ±0.023	0.850 ±0.017	0.930 0.007

Results (II)

Table: Improvement achieved by *SoftId* classifier compared to *OSVM*. For a performance measure *P*, the table depicts the value *P(SoftId)-P(OSVM)*.

No. of original authors	N-grams	TF-IDF representation							LSI representation						
		Acc	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*	*Acc*	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*
3	**5-grams**	0.043	0.008	0.040	0.025	0.071	0.056	0.024	0.038	0.011	0..031	0.023	0.107	0.069	0.021
	6-grams	0.058	0.008	0.056	0.035	0.077	0.066	0.032	0.058	0.011	0.053	0.035	0.104	0.079	0.032
	8-grams	0.039	0.000	0.043	0.024	0.000	0.021	0.021	0.044	0.003	0.046	0.027	0.027	0.036	0.024
	10-grams	0.043	0.000	0.047	0.027	0.000	0.024	0.024	0.044	0.010	0.039	0.027	0.099	0.069	0.025
5	**5-grams**	0.072	0.022	0.060	0.042	0.196	0.128	0.041	0.077	0.042	0.045	0.044	0.388	0.217	0.044
	6-grams	0.075	0.015	0.070	0.045	0.129	0.100	0.042	0.085	0.021	0.075	0.051	0.185	0.130	0.048
	8-grams	0.075	0.000	0.083	0.046	-0.005	0.039	0.041	0.065	0.011	0.062	0.040	0.095	0.078	0.036
	10-grams	0.077	-0.001	0.086	0.048	-0.012	0.037	0.043	0.069	0.016	0.062	0.042	0.141	0.101	0.039
12	**5-grams**	0.072	0.031	0.050	0.041	**0.292**	**0.171**	**0.041**	0.085	0.063	0.032	0.047	**0.619**	**0.325**	**0.047**
	6-grams	0.077	0.031	0.057	0.045	0.282	0.169	**0.044**	0.071	0.043	0.039	0.041	0.398	0.218	0.041
	8-grams	0.058	0.018	0.048	0.034	0.155	0.102	0.033	0.078	0.027	0.061	0.046	0.241	0.151	0.044
	10-grams	0.059	0.011	0.056	0.037	0.087	0.071	0.034	0.071	0.020	0.061	0.044	0.173	0.117	0.041

Outline

Conclusions

- the *SoftId* classifier successfully solves the software authorship attribution task in an open-set configuration
- *SoftId* outperforms the One-Class SVM classifier in an overwhelming majority of testing configurations with respect to all performance measures
- the textual representations used are relevant for distinguishing between authors
- **future work**: evaluate *SoftId* on data sets collected from software development teams

Bibliography

Mohammed Abuhamad, Ji-su Rhim, Tamer AbuHmed, Sana Ullah, Sanggil Kang, and DaeHun Nyang.
Code authorship identification using convolutional neural networks.
Future Generation Computer Systems, 95:104–115, 2019.

Sarkhan Badirli, Mary Borgo Ton, Abdulmecit Gungor, and Murat Dundar.
Open set authorship attribution toward demystifying Victorian periodicals.
Lecture Notes in Computer Science, 12824:221–235, 2021.

Veronika Bauer, Tobias Volke, and Sebastian Eder.
Comparing TF-IDF and LSI as IR technique in an approach for detecting semantic re-implementations in source code, 2015.

Project code Software Campus, TU Munchen, grant number 01IS12057.

Tom Fawcett.
An introduction to ROC analysis.
Pattern Recognition Letters, 27(8):861–874, 2006.

Google.
Google Code Jam Competition.
https://codingcompetitions.withgoogle.com/codejam.
Online; accessed 15 September 2021.

Vaibhavi Kalgutka, Ratinder Kaur, Hugo Gonzalez, Natalia Stakhanova, and Alina Matyukhina.
Code Authorship Attribution: Methods and Challenges.
ACM Computing Surveys, 52:1–36, 2020.

M. Koppel and J. Schler.
Authorship verification as a one-class classification problem.

In *Proceedings of the Twenty-first International Conference on Machine Learning*, pages 1–8, 2004.

J.I. Maletic and A. Marcus.
Using latent semantic analysis to identify similarities in source code to support program understanding.
In *Proceedings of 12th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2000*, pages 46–53, 2000.

Anand Satyam, Kumar Dawn Arnav, and Kumar Saha Sujan.
A statistical analysis approach to author identification using latent semantic analysis.
In *Conference and Labs of the Evaluation Forum (CLEF)*, pages 1143–1147, 2014.