Who's this? Developer identification using IDE event data

John Wilkie, Ziad Al Halabi, Alperen Karaoglu, Jiafeng Liao, George Ndungu,
Chaiyong Ragkhitwetsagul, Matheus Paixao, Jens Krinke
CREST, University College London
London, UK

ABSTRACT

This paper presents a technique to identify a developer based on their IDE event data. We exploited the KaVE data set which recorded IDE activities from 85 developers with 11M events. We found that using an SVM with a linear kernel on raw event count outperformed *k*-NN in identifying developers with an accuracy of 0.52. Moreover, after setting the optimal number of events and sessions to train the classifier, we achieved a higher accuracy of 0.69 and 0.71 respectively. The findings shows that we can identify developers based on their IDE event data. The technique can be expanded further to group similar developers for IDE feature recommendations.

CCS CONCEPTS

• **Software and its engineering** → *Software notations and tools*; *Development frameworks and environments*;

KEYWORDS

Machine Learning, Integrated Development Environment, Recommendation Systems

ACM Reference Format:
John Wilkie, Ziad Al Halabi, Alperen Karaoglu, Jiafeng Liao, George Ndungu, Chaiyong Ragkhitwetsagul, Matheus Paixao, Jens Krinke. 2018. Who's this? Developer identification using IDE event data. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196398.3196461

1 INTRODUCTION

Developers' activities are recorded in their integrated development environment (IDE) sessions. This rich information, which captures the developers' behaviour in real time, can be used to identify a developer or group similar developers together. The identification and clustering of developers have two benefits. First, the IDE vendors can use this knowledge for feature recommendation in their IDEs. Visual Studio 2017 has redefined its setup process by allowing the user to choose which features are to be added after installing the core components [5]. This modularisation resulted in lower resources required by the IDE and faster start time, which enhanced the developers' productivity. Nevertheless, developers have to decide which features to install by themselves based on required tasks at hand or personal development styles. A feature can be recommended to a user if it is commonly installed among other users who share similar IDE usage patterns. Second, the developer classification enables IDE vendors to gain insights on which group of developers are using which IDE features and precisely target customers for beta testing of a new feature.

Our work is inspired by the work of Rosemblum et al. [7], which presented techniques to identify the author of program binaries and cluster programmers according to their style. Instead of analysing stylistic features extracted from program binaries like in their techniques, we exploited a rich information encoded in IDE event data and trained classifiers to match a programming session to a specific developer based on actions they perform in an IDE. We analysed the Enriched Event Streams (EES) [6], which are streams of recorded event data around a developer's actions within Visual Studio. We considered two types of classifiers: Support Vector Machines (SVM) [3] and k-Nearest Neighbours (*k*-NN) [10], and evaluated preprocessing steps to improve the accuracy of the classifiers by incorporating term frequency and inverse document frequency (tf–idf) to the input data.

The experiment shows that the SVM classifier gives a higher accuracy than *k*-NN when identifying developers based on their IDE event data. By adjusting the minimum number of events in a session and the minimum sessions per developers, we can achieve an accuracy of 0.71. This result shows that our technique can be used to identify developers based on their IDE event data, and may be further used to recommend similar IDE features to developers.

2 RELATED WORK

There are a number of existing studies on identifying and classifying developers and techniques to improve the classification performance. The majority of the approaches focus on analysing software artefacts rather than developers' behaviour. Caliskan-Islam et al. [1] aimed at de-anonymising source code authors by analysing coding styles based on lexical, layout, and syntactic features. The source code was presented as abstract syntax trees, and a tf–idf score was calculated for every node. This allowed fragments of code to be expressed as numerical vectors. A random forest ensemble classifier was applied with an accuracy of 0.94 when classifying 1,600 authors. Rosenblum et al. [7] used an SVM for author classification and a large margin nearest neighbors algorithm for clustering based on stylistic featured derived from program binaries. They achieved an accuracy of 0.77 when applied on a set of 20 authors, and ranked the correct author among the top five 94% of the time. Similar to our study, Dyke [4] utilised a data set of automatically captured developers' behaviour in an IDE to classify them. The technique identified novice developers, who required additional help, using data captured from 124 students. The author found that frequencies of editing, saving, code generation and testing of a program in non-debugging mode are good indicators of a student's success in an assignment.

Classification algorithms, which predict a class that an element belongs to, are commonly used tools for data mining. k-Nearest Neighbours (*k*-NN) and Support Vector Machine (SVM) are two widely used classifying algorithms. In order to classify an element, the *k*-NN algorithm calculates the distance between a test element and all the elements in its training set and finds a group of *k* elements that are closest to it. The test element is then classified based on the majority class of its *k* nearest neighbours. *k*-NN is well suited for cases where elements can have multiple class labels [10]. A more sophisticated classification scheme is SVM, which finds the best classification function to differentiate between the different classes of elements of the training set. The SVM algorithm identifies the optimal hyper-plane that will segregate the training set classes. SVM are considered one of the most robust and accurate classifying algorithms available. A major advantage of using SVM algorithms is that they are insensitive to the number of dimensions [10].

Term frequency and inverse document frequency (tf–idf) [8] is a vector-based weighting scheme often used in information retrieval. Words that frequently occur in a specific document but rarely in other documents in the corpus have a higher tf–idf score compared to frequent words that occur in most of the documents. The aim of applying tf–idf is to identify important words in a document as not all words are equally important in characterising a document. Applying tf–idf as a data pre-processing step preceding a classification process has been successfully used in previous studies. For document categorisation, Trstenjak et al. [9] used tf–idf with a *k*-NN classifier to classify documents into 4 classes (sport, politics, finance and daily news). They classified 500 documents of varying lengths and obtained an accuracy from 0.65 to 0.92. Chen et al. [2] compared the use of tf–idf and a simple frequency based scheme as pre-processing steps to training an SVM, and an Artificial Neural Network (ANN) model to detect potential systems intrusions. They found that an SVM with tf-idf achieved the best performance.

3 RESEARCH METHODOLOGY

In this section, we briefly discuss the data set, the data preparation process and the classification methods used in our study.

3.1 The KaVE Data Set

The KaVE data set [6] was created using the FeedBaG++ tool to capture developers' behaviour within an IDE and transform them into Enriched Event Streams. The data set contained events from 85 developers from a variety of backgrounds (industry, research, students and hobbyists) and in total contains 11M events. A session contains multiple events from a specific developer that occurred in a single day. An event is added to an event stream when a developer interacts with an IDE, for example by clicking the build button. Only developers with more than a single session were selected. This gave a total of 3,459 sessions and 82 developers. The analysis of the number of sessions per developer and events per session is depicted in Figure 1. The mean, and median sessions per developer was 42.2 and 27.5 respectively. The lowest number of sessions for a developer was 2 and the highest was 207. The total number of events across all sessions was 11,027,743. The mean number of events in a session was 3,188 and the median was 277, which indicates a large range in the number of events in sessions. The lowest number of events in a session was 1 and the highest was 353,585.

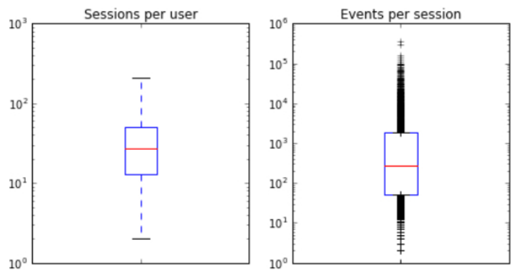

Figure 1: Boxplots showing the distribution of sessions per developer and events per session

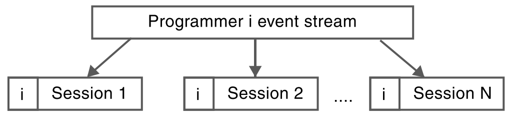

Figure 2: Initial stage in splitting developer event stream

3.2 Data Preparation

In order to apply an SVM and a *k*-NN classifier to the KaVE data set, we performed a data pre-processing step to create an input data set that is suitable for the classifiers. The initial stage, as depicted in Figure 2, consisted of splitting each developer's event streams into a set of event streams per session and labelling each session's event streams with a unique developer ID to identify the developer whom the session belonged to. For each session, a count was carried out to sum the number of occurrences of each event type within that session. We created three levels of Event Granularity (EG). These three levels of EG were used for representing the types of events being counted. The coarse-grained EG (CEG) consists of 15 event types (activity, command, completion, build, debugger, document, edit, find, IDE state, solution, window, version control, navigation, system, and test run) used in the EES as shown in Figure 3 (error, info and user profile events were not included because the error and info events do not occur in the data set and the user profile event was not considered an IDE event). The medium-grained EG (MEG) further divided the CEGs into smaller sub groups of events. For example, a build event was split into: build, build all, rebuild all and unknown. This resulted in 49 different MEG types. Lastly, the fine-grained level EG (FEG) made use of the 49 MEG types and also subdivided the command event into another 42 new types of events, giving 91 FEG types in total. Each session was represented by a concatenation of the three vectors, one for each of the three levels of granularity, as shown in Table 1[1].

Besides the three granularity levels, we evaluated two types of input vector values to the classifiers in this study: raw event

[1]A complete information of our event categorisation can be found from the study website: https://ziad-alhalabi.github.io/msr-paper-2018/

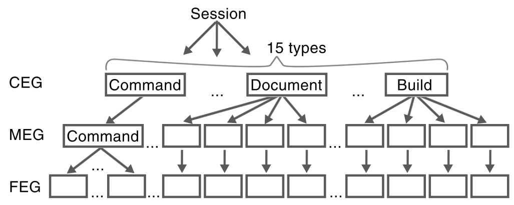

Figure 3: Event granularities

Table 1: A CEG vector with raw counts of 15 event types

CEG				
Completion	Build	Edit	...	Debugger
12	6	21	...	56

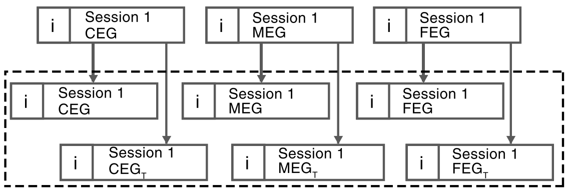

Figure 4: The six vectors representing a session

counts and tf–idf. For each session, we first created three session vectors (CEG, MEG, and FEG) with raw event counts and derived three vectors of tf–idf from them. This resulted in 6 vectors for each session CEG, CEG with tf–idf (CEG$_T$), MEG, MEG with tf–idf (MEG$_T$), FEG and FEG with tf–idf (FEG$_T$) as shown in Figure 4.

Each session vector was then grouped together with all other session vectors of the same EG, and grouped based on whether tf–idf had been applied. For example, all session vectors which had been generated using CEG$_T$ were grouped together into one set. These session groups were used as the training data for classifiers.

3.3 Programmer Classification: *k*-NN and SVM

The classification uses the preprocessed vector data. For each developer, we randomly selected a fixed number of sessions from all their sessions. Then, the selected sessions were split into training and testing sets. The training set utilised 80% of the original data, while the remaining 20% was assigned as the testing set. The training set was used with *k*-fold cross validation to tune the parameters for the two classifiers. In the case of *k*-NN, we tuned the threshold for distance measure and the number *n* for neighbours. In the case of an SVM, the training data is used for selecting the kernel and tuning the penalty of the error term, which resulted in a linear kernel to be selected. With the parameter selection and tuning, we trained the classifiers using the training data set, then the trained classifiers were used to predict the most likely developer for each session in the testing data set. These predictions were then compared to the true developer of the session and the accuracy was calculated as follows:

$$\text{accuracy} = \frac{\text{no. of correctly classified sessions}}{\text{no. of all sessions in test data}}$$

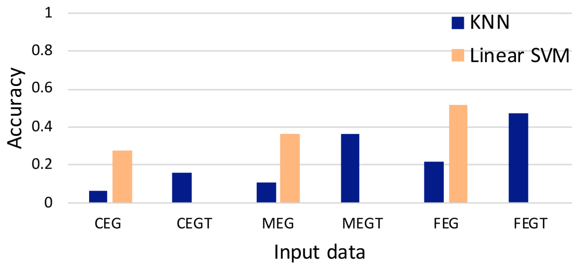

Figure 5: Classification performance on six input data types

3.4 Research Questions

We performed an experiment to answer the following research questions.

RQ1: A Comparison of SVN to *k*-NN *Which classifier offers the best performance on identifying developers based on their IDE events?*

RQ2: Minimum events per session *How many events per session give the highest classification accuracy?*

RQ3: Minimum sessions per developer *How many sessions per developer give the highest classification accuracy?*

4 RESULTS

RQ1: A Comparison of SVN to k-NN. This RQ compares the SVM classification performance compared to *k*-NN for six different event granularities and vector values: CEG, CEG$_T$, MEG, MEG$_T$, FEG, FEG$_T$ data sets. We filtered the 6 data sets to only include rich sessions with at least 500 events (42% of all the sessions) with the aim of removing sessions that contained too few events to characterise a developer. This assumption was then explored in RQ2. From this data, developers with at least 10 sessions were selected (40 out of 82 developers), and then for each developer 10 sessions were randomly selected so that each developer has the same number of sessions in order to avoid an evaluation bias. This resulted in 40 developers with a total of 400 sessions. Each classifier was trained using 80% of the sessions, and the accuracy was measured by classifying the remaining 20% of sessions. The results are shown in Figure 5. The highest accuracy of 0.52 was achieved using the SVM classifier on FEG. The lowest accuracy (close to zero) occurred using the SVM on all data sets with tf–idf. In contrast, the *k*-NN classifier performed more accurately on the data sets with tf–idf, i.e. CEG$_T$, MEG$_T$, FEG$_T$. The highest accuracy achieved using the *k*-NN classifier was 0.475 on FEG$_T$ data set. Both classifiers performed best, with or without tf–idf, on the FEGs, followed by MEGs, and CEGs respectively.

To answer RQ1, the findings indicate that an SVM with a linear kernel is a more appropriate classifier for IDE event data than *k*-NN, since SVM classifiers often result in a higher accuracy when used on high dimensional but sparse data [10]. It was found to be detrimental to apply tf–idf to session vectors before training the SVM. This result is contrary to the findings of Chen et al. [2]. However their application was detecting intrusions to a system and not classifying sessions to developers. We leave a detailed investigation of this issue as future work.

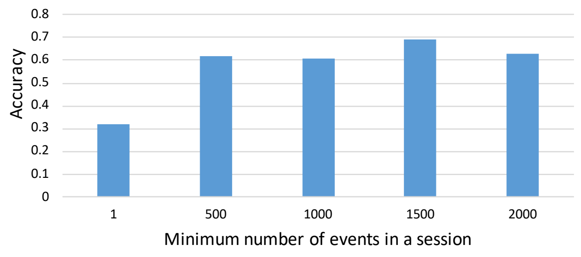

Figure 6: Minimum number of events vs. accuracy

RQ2: Minimum events per session. Five thresholds were considered starting with zero events (all sessions included) with an increasing step of 500 up to a threshold of 2,000 events. This was done using the best performing data granularity and classifier in RQ1. FEG data was selected to train an SVM classifier on 10 sessions for each developer. Similar to RQ1, 10 sessions for each developer were randomly selected from all of the developer's sessions. To control the number of developers across the five threshold values, we selected the 23 developers that had at least 10 sessions with at least 2,000 events. The results of the changing thresholds on the accuracy of the classifier can be seen in Figure 6. The highest accuracy of 0.69 was achieved when the threshold was set to 1,500 followed by 0.63 using a threshold of 2,000 and 0.62 using a threshold of 500. The increased accuracy for the threshold of 500 in comparison to the findings of RQ1 is due to having fewer developers to classify a session to and thus making the classification task simpler.

RQ3: Minimum sessions per developer. The chosen values for the number of sessions included 5, 10, 15, 20 and 25. Again, we tested using an SVM classifier with FEG data, with sessions of at least 500 events. We obtained 20 developers that had at least 25 sessions to control the number of developers. The classification accuracy can be seen in Figure 7. The highest accuracy of 0.71 was achieved when the threshold was set to 20 sessions per developer, followed by 25 and 15 sessions.

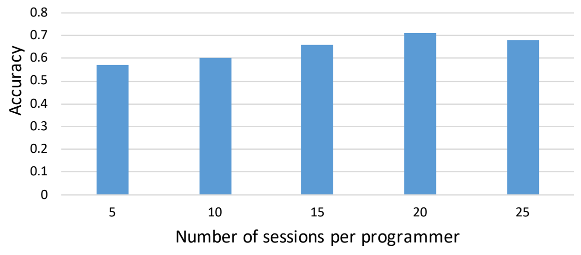

Figure 7: Number of sessions vs. accuracy

5 DISCUSSION AND THREATS TO VALIDITY

The results indicate that during pre-processing of the input data, setting a minimum number of sessions per developer and a minimum number of events per session is beneficial. The findings of this paper suggest that using 20 sessions per developer and removing sessions with less than 1,500 events offer the most accurate classification.

The investigation outlined in this paper used a small data set for the type of techniques being applied. The results are considered as a foundation for a more comprehensive data set, which we leave for future work. A larger set of event stream data would be beneficial to increase the validity of the findings in this paper and potentially improving the classification accuracy.

The work done in this paper only considers the occurrence of each event, and ignores other features which may help to classify a session to an individual developer. Example features that may increase accuracy could be the rate at which the developer types, patterns in their variable names or indentation rules they follow. Additionally, only single events were considered when building the session vectors. The accuracy may be improved if we consider a sequence of multiple events. The possibility of combining classifiers for the three different event granularities into a single classifier could also be evaluated. One very important concern is that ethical considerations would be required if this type of classification were possible. Moreover, The results in the paper may be affected by the problems in the KaVE event stream data that duplicated events were generated for some event streams, which were recently discovered[2].

6 CONCLUSIONS

This paper presents a classification of programming sessions to a developer using events that occurred in an IDE event stream. The highest accuracy was achieved using an SVM classifier with a linear kernel on raw event count data. Applying tf–idf to raw event counts was found to be detrimental to the accuracy of the SVM classifier, but beneficial to *k*-NN. The threshold analysis shows that the SVM classifier can be tuned to gain the optimal performance using a minimum of 1,500 events per session and 20 sessions per developer. The findings show that one can identify developers based on their IDE event data. Moreover, the results can be used as a foundation for recommending IDE features based on developers with similar behaviour.

REFERENCES

[1] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing Programmers via Code Stylometry. In *USENIX Security '15*.
[2] Wun-Hwa Chen, Sheng-Hsun Hsu, and Hwang-Pin Shen. 2005. Application of SVM and ANN for Intrusion Detection. *Comput. Oper. Res.* 32, 10 (Oct 2005).
[3] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning* 20, 3 (Sep 1995).
[4] Gregory Dyke. 2011. Which Aspects of Novice Programmers' Usage of an IDE Predict Learning Outcomes. In *SIGCSE '11*.
[5] Microsoft. 2017. What's New in Visual Studio 2017. (2017). https://docs.microsoft.com/en-us/visualstudio/ide/whats-new-in-visual-studio
[6] Sebastian Proksch, Sven Amann, and Sarah Nadi. 2018. Enriched Event Streams: A General Dataset for Empirical Studies on In-IDE Activities of Software Developers. In *MSR '2018 Mining Challenge Proposal.*
[7] Nathan Rosenblum, Xiaojin Zhu, and Barton P Miller. 2011. Who wrote this code? identifying the authors of program binaries. In *ESORICS '11*. Springer.
[8] Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA.
[9] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. 2014. KNN with TF-IDF based Framework for Text Categorization. *Procedia Engineering* 69 (2014).
[10] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. 2008. Top 10 algorithms in data mining. *Knowledge and Information Systems* 14, 1 (Jan 2008).

[2]https://groups.google.com/forum/#!topic/kave-users/LUM3-vkdDlE