Privacy versus Information in Keystroke Latency Data

Juho Leinonen

M. Sc. Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, November 25, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Laitos — Institution — Department Department of Computer Science
Tekijä — Författare — Author Juho Leinonen	
Työn nimi — Arbetets titel — Title Privacy versus Information in Keystroke Latency Data	
Oppiaine — Läroämne — Subject Computer Science	

Työn laji — Arbetets art — Level M. Sc. Thesis	Aika — Datum — Month and year November 25, 2016	Sivumäärä — Sidoantal — Number of pages 59

Tiivistelmä — Referat — Abstract

The computer science education research field studies how students learn computer science related concepts such as programming and algorithms. One of the major goals of the field is to help students learn CS concepts that are often difficult to grasp because students rarely encounter them in primary or secondary education. In order to help struggling students, information on the learning process of students has to be collected. In many introductory programming courses process data is automatically collected in the form of source code snapshots. Source code snapshots usually include at least the source code of the student's program and a timestamp. Studies ranging from identifying at-risk students to inferring programming experience and topic knowledge have been conducted using source code snapshots.

However, replicating source code snapshot -based studies is currently hard as data is rarely shared due to privacy concerns. Source code snapshot data often includes many attributes that can be used for identification, for example the name of the student or the student number. There can even be hidden identifiers in the data that can be used for identification even if obvious identifiers are removed. For example, keystroke data from source code snapshots can be used for identification based on the distinct typing profiles of students. Hence, simply removing explicit identifiers such as names and student numbers is not enough to protect the privacy of the users who have supplied the data. At the same time, removing all keystroke data would decrease the value of the data significantly and possibly preclude replication studies.

In this work, we investigate how keystroke data from a programming context could be modified to prevent keystroke latency -based identification whilst still retaining valuable information in the data. This study is the first step in enabling the sharing of anonymized source code snapshots. We investigate the degree of anonymization required to make identification of students based on their typing patterns unreliable. Then, we study whether the modified keystroke data can still be used to infer the programming experience of the students as a case study of whether the anonymized typing patterns have retained at least some informative value. We show that it is possible to modify data so that keystroke latency -based identification is no longer accurate, but the programming experience of the students can still be inferred, i.e. the data still has value to researchers.

Avainsanat — Nyckelord — Keywords
Keystroke identification, Data privacy, Data anonymization, Source code snapshots

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Nowadays, a lot of data is shared openly for replication studies and novel analysis on existing data [11, 18, 39]. Still, privacy issues often prevent companies, governments, and (educational) institutions from sharing the data that they have collected [23]. For example, on many computer science courses, very fine-grained data from the students' working process is collected in the form of source code snapshots. Such snapshots have been used for studying varying topics such as detecting struggling students [1], inferring programming experience [42], and studying pausing behavior [43]. Replicating keystroke latency -based studies is currently hard as sharing non-anonymized data that could be used to identify individuals would violate the privacy of the users or parties from which the data has been collected. Anonymizing data by simply removing parts of the data – attributes – may not be sufficient as hidden factors that can be used to identify individuals may exist.

For example, the online movie streaming platform Netflix held an open competition to improve the accuracy of their recommendations to users. To help competitors develop their algorithms, Netflix released an anonymized data set containing movie ratings given by users. Narayanan and Shmatikov were, however, able to link the data to another data set collected from the online movie database Internet Movie Database and identify individual users from the data [55]. Based on the supposedly anonymous ratings that individuals had given, information such as political beliefs could, in the end, also be inferred.

Attributes that are not identifiers by themselves, but can be used for identification together with other attributes are called quasi-identifiers [23]. For example, Daries et al. [17] studied anonymization of MOOC data from a social science perspective, and defined the country, gender, age and level of education of a participant as quasi-identifiers. Similarly, keystroke timings found in programming snapshots are quasi-identifiers: a single keystroke timing does not reveal the identity of the typist, but together the timings can be used to construct a typing profile that can be used for identification [20, 24, 36, 47, 52]. For example, Longi et al. [47] have showed that individual programmers can be identified from source code snapshots based on the times that the programmers take to move from one key to another, i.e. the typing pattern.

It is rare to include keystroke data in open data sets. While source code snapshot data is publicly available by, for example, the Blackbox-project [11], the data does not include keystroke level data. Thus, keystroke timing -based studies (e.g. [9, 20, 42, 68]) are presently hard to replicate because such data is rarely collected and available. This has been acknowledged as a problem and there seems to be pressure (and a trend) for publishing more fine-grained learning data than what is available today [33].

Al-Zubidy et al. note that replication studies are essential for theory building and are therefore concerned about the lack of replication studies in the computer science education field [3].

Leinonen et al. [42] have shown that programming experience can be inferred from keystroke timings to a degree. They classified students into experienced and novice programmers with different machine learning methods and were able to achieve classification accuracies of up to 77%, which were significantly higher than classification accuracy with the majority classifier which had an accuracy of 58.8%. This indicates that typing profiles contain information on programming experience, and indeed, Leinonen et al. noticed that the features – average latencies between character pairs – with the most predictive power were related to programming and that the results were likely explained by experienced programmers being able to type programming related keywords and symbols more quickly than novice programmers.

Daries et al. [17] showed that in a social science context, the value of data can degrade significantly in the anonymization process – results on anonymized data differ from results on non-anonymized data. In this work, we study whether there is a similar effect in anonymizing source code snapshot data. More specifically, we investigate whether keystroke timing data in source code snapshots can be modified in a way that prevents typing pattern -based identification, whilst other valuable information can still be inferred from the anonymized keystroke timing data. We conduct a case study where programming experience is the valuable information we wish to be able to infer from anonymized keystroke timing data.

The novel contributions of this work are as follows. We conduct experiments using two anonymization procedures and compare identification accuracies with different degrees of anonymization. Furthermore, we seek to find a balance where programmers could not be identified based on keystroke timings but programming experience could still be inferred. Being able to infer programming experience but not the individuals would suggest that there is value for researchers in the data, while the privacy of the individuals would be preserved. This is a step towards releasing fine-grained source code snapshot data openly to others.

We focus on preventing identification based on keystroke timings, which does not guarantee that data is anonymous since identification could also be possible from other identifiers found in keystroke data such as text content (variable names, class names, etc.). However, being able to modify keystroke timings so that they can not be used for identification would remove a quasi-identifier from the data, which would maintain the possibility that anonymized keystroke timings could be included in open data sets and used for research.

This work is organized as follows. First, in Section 2, background on source code snapshot analysis, keystroke identification, inference based on keystroke timings, and data anonymization techniques are presented. Second, in Section 3, the research design of this work is outlined, including the research questions and the context of the data that is used in the study. Then, in Section 4, we detail our research methodology, followed by the experiments we conduct to answer our research questions and the results of the experiments in Section 5. In Section 6, the results of this work and their consequences are discussed. Finally, Section 7 concludes this work and presents future avenues of research.

This thesis builds on previously published research of the author and his colleagues [41, 42, 47]. Text that has been written by the author for the original articles is used in some parts of this work, mainly in the background, research design, and methodology sections (Sections 2, 3, and 4).

2 Background

Here, we visit four streams of related work. First, background on source code snapshot analysis is covered, followed by discussion on articles where keystroke timings have been used for inferring the identity of a user. Then, we review articles related to inferring other information in addition to identity from keystroke timings, and finally, we visit data anonymization focusing on different anonymization techniques.

2.1 Source Code Snapshot Analysis

Recording data of students learning processes is increasingly popular. On many introductory programming courses, such as those provided at the University of Helsinki, the students' complete working process is recorded. For example, most of the students in the "Introduction to Programming" and "Advanced Course in Programming" use a plugin called TestMyCode [71] that records snapshots of the students' progress on assignments. The data is very fine-grained as it includes every keystroke the students type in the programming environment. In addition to snapshots, usage data is being collected from other learning sources such as online course materials.

A recent literature review that analyzed data collection in the context of computing education found that only 76 out of over 3500 articles included automatic programming process data gathering [33]. The Blackbox-project [11] provides researchers access to source code snapshot data collected from the BlueJ development environment. However, the data does not contain keystroke level data and thus can not be used for replicating keystroke level studies. Replication and validation studies would be essential for confirming studies can generalize from one context to another, and that context specific factors are not the only contributing factors to phenomena observed in source code snapshot -based studies.

When source code snapshots are used for research, they are often preprocessed, e.g. converted or aggregated to a more abstract or coarse level [31]. This is one of the main benefits of having very fine-grained data as in contrast, more coarse-grained data can not be converted to fine-grained data. For example, if source code snapshots are only collected when students run their code, data in between running the code is lost. In this case, keystroke level studies could not be conducted with the data as it is unlikely the students run their code after typing every keystroke, and keystroke information could not be derived post hoc.

For example, Rivers and Koedinger have developed a programming tutor that automatically generates hints based on states which are built based on programming code [61,62]. In this case, they first convert the code into states that are irrespective of differences in code syntax. For example, the variable and class names in the code are renamed deterministically so that the state of the code is the same regardless of how students name the variables or classes. Then, the transitions between these states are used to see whether the student is moving towards or away from the states that represent the possible correct solutions to the problem. The states and transitions built based on source code snapshots can be seen as the programming paths the students take while completing exercises. Other research has studied programming paths to analyze the students' working process: Hosseini et al. found that students build their programs incrementally in small steps [30].

Leppänen et al. [43, 44] have also studied students' working processes from source code snapshot data. They analyzed the working process of students focusing especially on the pauses the students take and how students space out their work, i.e. how many days do they work on the course assignments. They found that students work on average three days a week on the exercises. Those who got more points in the exam worked on less days on average, which was quite surprising considering that it has been previously shown that spacing out work can be beneficial for learning, at least in the context of recalling information [12, 13, 19]. In addition to studying spacing out work, they studied what kind of pauses students take during exercises and how those spaces correlate with the exam points. The found that students who take a lot of short pauses that last from ten seconds to four minutes perform worse in the exam. They hypothesize that this is due to either students multitasking when they are completing the exercises, resulting in a lot of task switching, which has been shown to be detrimental for learning [54]. Another hypothesis they postulate is simply that lower performing students have to go back to the material more to refresh unclear concepts. Leppänen et al. [43, 44] used snapshots from two different introductory programming courses held at the University of Helsinki – one of the data sets was the same one that is used in the experiments in this work. This is a good example of the versatility of source code snapshots: there are a multitude of research avenues that can use source code snapshots for studies.

Another avenue of research that has utilized source code snapshots is predicting students programming performance [69, 74] and automatically identifying struggling students [2]. The Error Quotient (EQ) -algorithm [34, 63] is developed for predicting students' success based on source code snapshots. It analyzes whether successive source code snapshots compile, basing its predictions on the relative amount of compiling and non-compiling snapshots. Watson et al. [74] developed an algorithm called Watwin that can be used to predict students' success based on source code snapshots. They developed their algorithm as the EQ-algorithm by Jadud et al. [34] is not optimal in all contexts [59], for example when the students are using an integrated development environment where non-compiling code is visualized and easily corrected by the students. The Watwin-algorithm is based on the time it takes for a student to resolve problems compared to other students who have completed the problem. Ahadi et al. [2] studied both the EQ [34] and Watwin [74] -algorithms – in addition to other machine learning methods – for identifying struggling students, but found that both EQ and Watwin performed poorly with their data. However, machine learning methods such as the Random Forest classifier were able to achieve better results. They speculate that EQ's and Watwin's poor performance was due to the fact that Ahadi et al.'s study had less data than other studies where EQ and Watwin have performed better.

2.2 Keystroke Latencies and Identity

In this subsection, we review previous studies on authentication and identification in the context of keystroke latencies, how the environment of the typist can affect identification accuracy, and lastly look at few studies on identification in online exams, and identification based on source code snapshots in greater detail.

2.2.1 Authentication and Identification

Information recorded from typing, such as the duration of keystrokes, pressure of keystrokes, and keystroke latencies, has been used for identification purposes [20, 24, 36, 47, 51, 52, 58]. Being able identify people based solely on their typing patterns can be useful for both companies and consumers using online services. Typing patterns can, for example, be used as an additional layer of security in addition to the traditional username and password combination in sensitive applications. This way, only knowing the password is not enough for an impostor to gain access to the service. Similarly, typing patterns can be used for plagiarism detection in online exams [41, 48]. Typing patterns are a biometric trait and as such can be used for similar purposes as other biometric traits such as the iris. The advantage over other biometric traits is that a typing pattern can be observed without specialized hardware, a simple keyboards suffices. In contrast, using the iris or fingerprints requires a specialized scanner.

Intuitively, it is understandable that typing patterns can be used for identification. For example, a programmer who uses the computer daily for hours will likely be a faster typist compared to someone who seldom uses the computer. However, simply comparing typing speeds in general has not yielded very accurate results (e.g. in [47]). Thus, more detailed typing profiles are often used. These more detailed profiles can include, for example, average latency of the typist before a specific letter on the keyboard. Even more detailed profiles might take into account latencies between any two specific characters or *digraphs*. Similarly, the latencies between three specific keys, *trigraphs*, could be used. For example, in the word *true*, there are three digraphs – *tr*, *ru*, and *ue* – and two trigraphs –

tru and *rue*. Sometimes, even longer character sequences are included. For example, Dowland and Furnell [20] included the most common 200 English words in their analysis. In their research, the best results were achieved with digraphs. Indeed, digraphs have been used extensively [20, 24, 47, 52]. In addition to latencies between keys, for example the time a key is held down can be considered in typing profiles. Killourhy and Maxion noticed that including hold timings in typing profiles improved the identification results [38].

2.2.2 Effect of Environment on Identification Accuracy

A factor that can affect identification results significantly is the environment of the typist and what they are writing. A lot of research has gone into studying how identification accuracies vary between users typing a transcribed text such as their password or other predetermined text and users typing freely whatever they come up with. Of these, transcribed text has been studied more [7,15,27,35,76], most likely due to the traditional authentication aspect of keystroke analysis. More recently, the focus has shifted to free text [7, 25, 47, 50]. The results have varied in the studies where the difference between free and transcribed texts have been examined explicitly. In a study conducted by Monrose and Rubin [52], identification results were significantly better when using transcribed text compared to free text with accuracies of 79% and 21% respectively. Their hypothesis for the significant difference is that with transcribed text, the typist does not need to stop and think about what they are going to write, but can just type whatever text they are supposed to write.

However, other studies have found free text to perform as well as transcribed text, or at least close to it. Killourhy and Maxion [38] conducted a study where twenty subjects had to write both free and transcribed text. Using lowercase digraph latencies and key hold timings, they found that results varied and neither transcribed nor free text could outperform the other in all cases. Similarly, Villani et al. [72] found that with 36 subjects, identification accuracies with transcribed text were only marginally better with transcribed text compared to free text.

Gunetti and Picardi [25] note that different keyboards could possibly affect keystroke identification accuracy. Villani et al. [72] studied the effect of the keyboard on identification accuracy. They compared laptop and desktop keyboards and found that if the subjects changed keyboards between the training and testing of the identification model, the results were significantly lower (around 60% accuracy) compared to when they used the same keyboard for both training and testing (around 98% accuracy). There were no noticeable differences between using only laptop or only desktop keyboards. However, when the data was mixed, i.e. both laptop and desktop keyboards were used in both training and testing, identification was possible with the same high accuracy of around 99%.

2.2.3 Identification in Online Exams

In traditional programming courses, students have usually been at least partly graded using pen and paper exams. One of the problems related to such exams is that they only partially connect to the practice conducted within such courses. Testing students in a more practical environment has been constrained due to the limited resources that are needed, for example, for authentication.

Previous work by Bennedsen and Caspersen [5] argues strongly for having machine examination on introductory programming courses. However, a big limitation for having machine examinations is the cost of overseeing students taking the exam. Leinonen et al. [41] have shown that it is possible to identify students in a machine examination based on their typing profiles, which means that the cost of machine examinations could be alleviated by having the students complete the exam remotely on their own devices, since cheating students could be identified based on their typing patterns. However, condemning a student for cheating solely based on their typing profile is not advisable, since there could be other factors that affect typing such as exam stress or a broken arm. Nevertheless, what could be done is that a flag could be raised in situations where the student is suspected of cheating based on their typing, and further analysis is performed manually. A limitation of their approach is that keystroke analysis can only identify cases where a student has someone else complete the exam for them, but not cases where the whole course is taken by someone else than the student. Since it is only possible to observe typing, a student could cheat by having a friend help them during the exam, but do all the typing themselves. However, at the same time, such behavior might likely also influence the typing patterns, which would be noticeable when typing profiles in the exercises and the exam are analyzed in the same way as changing from transcribed to free-text does [52].

Keystroke analysis has been applied successfully for identifying students in online exams [41, 51, 65]. Using data from 30 students taking examinations in a business school, Monaco et al. were able to correctly identify all the students [51]. Likewise, Leinonen et al. [41] were able to identify a large portion of the students in programming exams where students code on a computer. They showed that students can be identified quite reliably in both controlled and uncontrolled exam environments. In the controlled exam, the students were in a computer lab at the university and in the uncontrolled exam they could be in whatever setting they found most comfortable, e.g. at home. However, they note that the identification accuracy is significantly lower than the accuracy achieved when students are identified on the last week of the course using the first six weeks as training data.

They found that using data from an exam does not perform as well as using a data set consisting of one week's worth of programming especially when identification is required to be exact. When the student is only required to be close enough, for example within the ten closest training set samples, identification accuracies of over 95% were observed.

2.2.4 Identification from Source Code Snapshots

A study by Longi et al. [47] shows that the identity of programmers can be detected from keystroke data recorded during programming sessions. They used data from two programming courses held at the University of Helsinki in the fall of 2014. One of the courses is the same as the course from which data is used in the experiments in this work. They studied how the amount of data affects identification accuracy with two experiments. In the first experiment, they increased the size of the training set week at a time and compared identification accuracies of successive experiments. They found that while exact identification accuracy rose from around 78% to over 95% when the size of the training set was increased from a single week to include six weeks of data, they were able to achieve an identification accuracy of over 95% when the student was allowed to be within the five closest training set samples to be considered correctly identified already when using only a single week as the training set. This means that already with a single week sized training set, possible impostors or cheaters can be identified quite well.

In addition to studying identification within a single course, Longi et al. [47] studied identification from one course to another. Being able to identify students across courses would indicate that releasing keystroke latency data poses privacy issues. For example, if a data set with typing information is released, it could be connected to a data set with typing information from another context. They found that students could be identified with extremely good accuracy when all data from the courses was used – only 2 out of the 145 students were not correctly identified, i.e. the identification accuracy was 98.6%. The identification in this case was exact, i.e. a correctly identified student had to be the closest match from the training set to be considered correctly identified. When this condition was relaxed from exact identification to allow the student to be within the closest five or ten typing samples in the training set, only a single student was not identified correctly. Longi et al. note that keystroke latency -based identification and authentication are especially convenient for long-distance courses such as Massive Online Open Courses (MOOCs) as they are irrespective of location and thus perfect for distance learning. The MOOC platform Coursera is already using keystroke identification as they collect typing samples from students seeking to acquire a verified certificate for completing a course [16].

One of the main ideas of Longi et al.'s identification model [47] is that exact identification is not always required. For example, in online exams it might be enough to only catch people whose typing profile is suspiciously different in the exam compared to the exercises as that might be a sign that someone else is completing the exam for them. Oppositely, exact identification is obviously required with authentication as there could be classified or personal information available in the system that requires authentication. Possibly due to this, the majority of previous work [6,7,15,24,35,48,50,53,53] on keystroke latency -based identification have focused on exact identification. Longi et al.'s model is inspired by the k-Nearest-Neighbors (kNN) classification algorithm. In kNN, a new sample is classified based on the classes of it's "neighbors". The neighbors are determined based on some distance function. A commonly used distance function is the euclidean distance, which is the one that Longi et al. used. Now, instead of trying to predict a sample's class, what is being predicted is the identity of the sample. Looking at the majority identity would not make sense as the samples are from different students, but instead in Longi et al.'s model the student is defined to be correctly identified if their sample is within the nearest k samples. Longi et al. call k the *acceptance threshold* as the value defines how many closest samples are accepted as correct. For example, with an acceptance threshold of five, a student is deemed to be correctly identified if any of the five closest training set samples comes from the student. So, in an exam scenario, we could build a typing profile based on the course assignments for each student and then build similar typing profiles in the exam. Next, we could calculate the distance from the students' assignment typing profiles to the students' exam typing profiles and check for each of the exam typing profiles whether the k closest typing profiles from the exercises include the student's exercise typing sample.

Longi et al.'s model was examined in more detail by Leinonen et al. [41] who found that an acceptance threshold of 5 seems to have about as good performance as a threshold of 10, and both perform significantly better than exact identification, i.e. using a threshold of one. There are a few exceptions though, which suggests that to be certain, a threshold of 10 should be used. With only 25 features and a threshold of 10, they were able to get over 95% identification accuracy with all data sets in their experiments. This means that in a real-world scenario, only 5% of the cases would be false positives, i.e. identifying a "cheating" student where there is no cheating. The acceptance threshold is still small enough to guarantee reasonably low false negative rate; for all contexts the probability of an impostor passing the identification test is below 10%.

2.3 Classification methods

In this subsection, we outline the classification methods that are used later in the experiments in this work. We will not cover these in great detail, but rather try to convey intuitive understanding behind the classifiers.

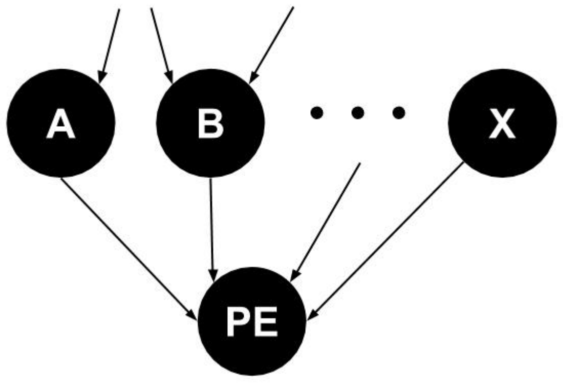

Figure 1: An example of a possible Bayes Net. Here A, B, X, and PE are states, and the edges represent dependencies. For example, PE is dependent on all A, B, and X. It could be dependent on other states as well, represented by the three dots.

2.3.1 Bayes Net

The Bayesian Network [4] (hereinafter shortened to Bayes Net) classifier is a probabilistic graphical model, which means that it can express random variables and their dependencies as a graph. An advantage of this is that the resulting graph is easily human-readable and can provide researchers knowledge on the underlying state of the system based on observed data. A Bayes Net contains states, transitions between the states, and conditional probabilities between the states. Figure 1 depicts an example of a Bayes Net graph.

The transitions between states in a Bayes Net are directed and cannot form cycles, i.e. a Bayes Net is a directed acyclic graph (DAG). The edges, i.e. transitions, express the dependencies between variables. For example, if there is an edge from a state A representing some random variable A' to a state B representing some other random variable B', then the value of B' depends on the value of A'. The values are truth values, i.e. either true or false.

Bayes Nets can be used for inferring the states of unobserved variables. The classifier can learn states, transitions, and conditional probabilities based on training data, and then use the learned information for inferring the value of a state based on incomplete new data. Here, incomplete means that there are states for which the value is not known in the new data.

For example, let's consider inferring the programming experience of a student. We can give Bayes Net information on students' programming background and their average typing latencies for selected character pairs, i.e. digraphs. Then, the classifier can learn, for example, that based on this data, it seems that the programming experience variable is dependent on the typing latencies. It could learn that students with previous programming experience type certain digraphs faster. After learning the dependencies between the latencies and programming experience, it could automatically infer the value of the programming experience random variable based on typing latencies. This would be useful when the real value for the variable is unknown. For example, we could teach the Bayes Net with values from a course where programming experience of the students is known based on a background survey, and infer the programming experience of students on another course where such information is not available. For example, in Figure 1, the state PE could be the programming experience of a student. If we do not know its true value, we could infer it based on the values of the states from which there is an edge to PE. The other states could be for example "average latency of the digraph *i+* is lower than 300 milliseconds".

2.3.2 Random Forest

Random forest classification [10] is based on decision trees. Decision trees are built based on data and can be used to predict the value of a random variable based on values of input variables. Decision trees contain rules which are used in predictions. Similar to Bayes Nets, decision trees can be represented by graphs and are quite easy for a human to understand. In a decision tree, the leafs of the tree depict the possible classes the predicted variable can be assigned to. To predict the value of a variable, a path is followed from the root of the tree until a leaf node is reached. The nodes in between the root and the leafs represent decisions and the next node is selected based on the values of the input variables.

Random forests are an extension of decision trees. In random forests, multiple decision trees are used instead of a single decision tree. This is done to decrease the chance of overfitting the training data, i.e. making predictions that are only good with the training data but which do not generalize to new data. The random forest will make the prediction that the majority of the decision trees that it contains make. For example, in classifying students' programming experience, if the random forest contains 200 decision trees, and 125 of them predict that the student has programming experience and 75 predict he does not, the random forest will make the majority prediction and predict that the student does have programming experience.

Random forests are optimized by using only a subset of the features in each tree [10]. This is done because ideally, the decision trees within a random forest do not correlate with one another. If they do correlate, then the more they correlate, the closer the prediction is to just a single decision tree making the prediction. For example, if there are multiple input variables that correlate a lot with the value that is being predicted, many of the decision trees might learn rules based on those strong predictors.

2.3.3 Majority Classifier

Compared to the previously covered Bayes Net and Random Forest classifiers, the majority classifier is very simple: it always categorizes a new sample to the class with the most observations in the training data. For example, if there are three classes A, B, and C, and there were 50, 25, and 25 observations for each class respectively in the training data, it would categorize each new sample to the A class, since most of the training data was from that class.

The usefulness of the majority classifier comes from using it as a baseline against which other classifiers can be evaluated. For example, let's consider inferring programming experience in a scenario where 75% of the students did not have programming experience and 25% of the students had some programming experience in the training data. Now, the majority classifier would always predict students to not have programming experience, since that was the majority class in the training data. If we observe new data with a similar distribution, i.e. again 75% of the students do not have programming experience and 25% have some, the majority classifier would achieve 75% classification accuracy. Obviously, when we use more sophisticated classifiers such as either Bayes Net or Random Forest, we would expect to achieve a higher than 75% classification accuracy in this scenario. Thus, the majority classifier is a good baseline classifier in scenarios where the expected size of the classes is not equal (e.g. our example where the other class was three times as big as the other). In this kind of scenarios, other classifiers should ideally achieve better classification accuracies compared to the majority classifier, and are not very good at inference even if the achieve seemingly good results, if those results are worse than the results of the majority classifier.

2.3.4 Assessing the Classifiers with Cross-validation

There are many ways of evaluating classifiers. For example, if there is enough data, some data can be left out of the training set and used as a test set later to assess the accuracy of the classifier. However, if the data set is not very large, leaving out data during training might not be optimal.

Cross-validation is a method of assessing classifiers that is most suitable for scenarios where there is not much data. In cross-validation, one data set is split into subsets that are iteratively used as either the training or the test sets. For example, in 10-fold cross-validation that is used later in the programming experience inference experiments in this work, the data is split into ten equal-sized subsets. Then, accuracy of the classifier is observed ten times, training the classifier with nine out of the ten subsets and using a single subset as the test set each time. Each subsets is used once as the test set. The average performance over these ten trials is calculated and assumed to give a good expectation of the generalizability of the classifier, i.e. the performance of the classifier with new data.

2.4 Inferring Information from Keystroke Timings

In addition to identification and authentication, keystroke timings can be used for inferring other information, such as demographic information [32], programming performance [42, 45, 46, 68], programming experience [42], and emotional states [9, 22, 73]. Idrus et al. have shown that demographic information such as age and gender can be inferred from typing data [32]. They were able to identify demographic information from both password typing and free text typing. This kind of information could be used in authentication: if the typing profile of the person who typed the password does not match the demographics of the user, entry to sensitive systems could be rejected. It could also be used for targeted advertising – a website could gather typing data and display advertisements relevant to the user based on demographics.

2.4.1 Programming Performance

There have been numerous studies showing a relationship between keystroke latencies and programming performance [42, 45, 46, 68]. Intuitively, an experienced programmer is likely to have a different typing pattern compared to a novice programmer as the type of text found in source code is typically quite different from other texts, e.g. essays. Especially certain programming related character pairs – digraphs – are most likely typed faster by an experienced programmer. Being able to infer the programming performance of a student could allow educators to identify struggling students based solely on their typing.

Thomas et al. have studied the relationship between keystroke latencies and programming performance [68]. They categorized digraphs into seven categories and calculated the mean latency by category. The categories were based on the type of the digraph: alphabetic characters, numerical characters, control keys, other keys, browsing keys, edge digraphs, and the H-category, which are digraphs where one of the keys is a browsing key. Their motivation behind the categorization is that similar digraphs likely have similar latencies; for example while there is probably a difference between the typing speed of alphabetic digraphs and browsing key digraphs, the latencies within these categories are probably close to each other. They analyzed the correlation between each of the categories and the programming performance of the students. They conducted two experiments: in the first one, programming performance was analyzed based on evaluation of code by experienced programmers and in the second, programming performance was measured by the score the students got in a lab exam and a written test. Their results indicate that there are strong statistically significant correlations between the mean latencies of some categories and programming performance. Especially numeric, edge, and the H-type digraphs had strong negative correlations with performance – this means that those who typed a digraph from these categories more quickly achieved better results in performance.

Recently, Leinonen et al. [42] partially replicated the study by Thomas et al. [68] by analyzing the relationship between programming performance and certain digraph latencies. They were only able to do a partial replication as information on control and browsing characters was not included in their data. Their results, while not as significant as Thomas et al.'s, were in line with the original study. Additionally, they found that the amount of data has a significant effect on being able to observe the correlations. In addition to replicating Thomas et al.'s study, they explored a number of machine learning methods to classify the students based on their exam performance. The Bayesian Network and Random Forest classifiers had the best average performance in the task, when evaluated with 10-fold cross-validation using classification accuracy and Matthews Correlation Coefficient [60] as performance measures. Matthews Correlation Coefficient is a measure used to assess the quality of predictions of a classifier. It is different from identification accuracy in that it takes into account also false values, i.e. false positives and false negatives, in its measures. The value is between -1 and 1, where 1 corresponds to perfect classification, -1 corresponds to perfect misclassification, i.e. all values are the opposite of their true values, and 0 corresponds to predictions that are essentially random.

2.4.2 Programming Experience

Programming experience has been shown to increase programming performance in some contexts [26, 75], but contrary results exist [8]. If programming performance is affected by programming experience, the correlation between typing patterns and programming performance could be just manifesting the programming experience of the students, i.e. the students might only be performing better due to having programming experience.

In addition to replicating the study by Thomas et al. [68], Leinonen et al. [42] described an experiment where they sought to identify students' past programming experience from keystroke latencies. They divided students into two populations: those with some programming experience and those with none. Then, they analyzed multiple different machine learning methods such as decision trees, Bayesian classifiers, and rule learner classifiers for classifying the students based on programming experience. Similar to the programming performance predictions, they found that the Bayes Net and Random Forest classifiers had the best average performance when evaluated with 10-fold cross-validation using classification accuracy and Matthews Correlation Coefficient [60] as quality measurements. They observed up to 77% classification accuracy and a Matthew's Correlation Coefficient of 0.54 in predicting whether a student had programmed previously or not. As an example, they showed that on average, experienced programmers move faster from the key *i* to the key *+*, i.e. experienced programmers type the digraph *i+*, faster than novice programmers. The probability distributions for this digraph's average latency for both experienced and novice programmers are depicted in Figure 2. Intuitively, this makes sense as the digraph *i+* is something programmers type often when incrementing an index variable, while it rarely occurs in regular text.

Leinonen et al. [42] also analyzed individual features, i.e. digraphs. They performed a qualitative analysis on a combination of the selected features that had the most predictive power over programming experience. Overall, special keys dominated the list, appearing in nearly 40% of the selected attributes. This is perhaps not surprising, as experienced programmers have probably used them before, and therefore are more familiar with their location on the keyboard. The most relevant digraphs for distinguishing between novices and non-novices are programming-related, and can be categorized into four categories: *common commands*, such as incrementing a variable (`i++`) or using the "OR"-operator (`||`), *common keywords* such as `true`, *transitions between characters that require the use of e.g. shift, ctrl or alt*, such as typing opening and closing brackets (`{}`), and the speed from backspace to various characters, which is likely related to rapid fixing of misspellings.

The difference between the speed with which a novice and non-novice moves typing specific character-combinations, here from typing the character `i` to `+`, is illustrated in Figure 2 and in Figure 3 for typing the character `|` twice. In the figures, the typing speeds for the digraph are normalized between 0 and 1 over all the students, and displayed as two probability density functions that depict the novices and non-novices.

Automatically inferring programming experience from typing could be useful for many purposes. For example, even if a course has a background survey, some students may choose to not answer. In addition, there could be cases where typing patterns for a course have been collected, but not programming experience information. In those cases, it could be beneficial to be able to at least make an informed guess on the programming experience of students.

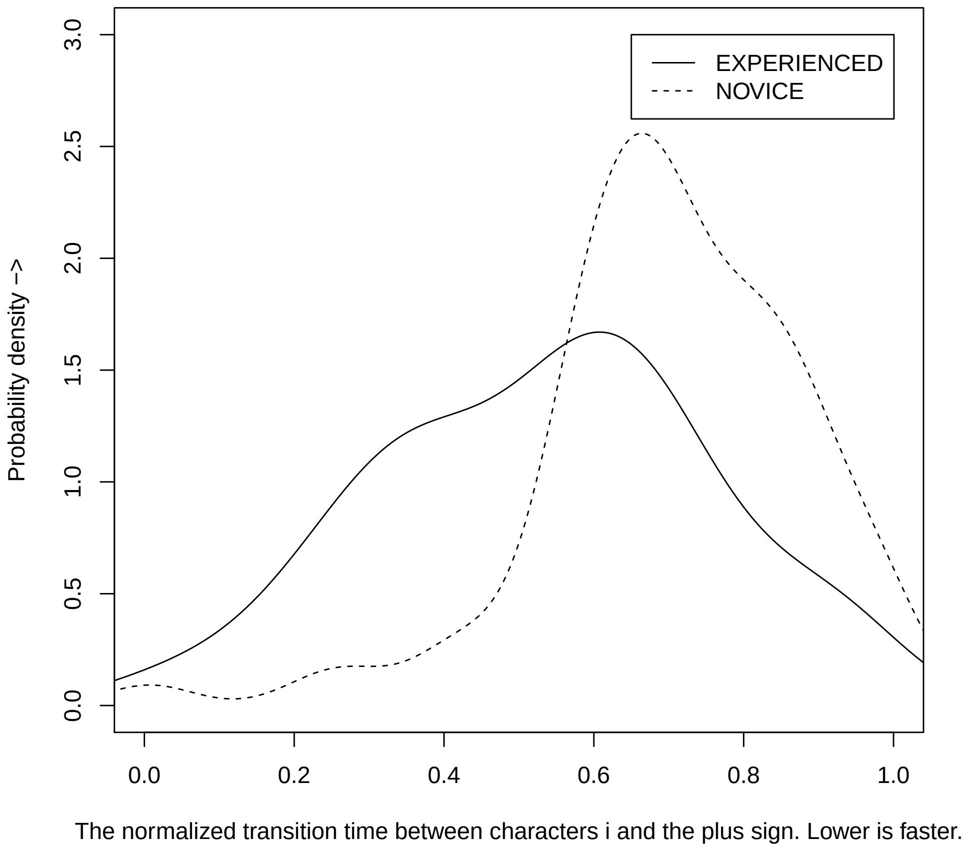

Figure 2: Smoothed probability density function of the times taken between pressing the characters i and + by novice and experienced programmers [42].

2.4.3 Emotional States

In addition to somewhat static attributes such as the identity, programming performance, and programming experience, keystroke analysis has been used to detect constantly changing features such as boredom and engagement [9], stress [73], and emotional states in general [22]. One of the advantages of measuring emotional states through keystrokes is that it is non-intrusive and cheap compared to other methods as noted in much of the research on keystrokes and emotions [9, 22, 73].

Bixler et al. [9] studied whether keystroke analysis could be used for detecting boredom and engagement during writing tasks. They conducted an experiment where students had to write three essays. They categorized stu-

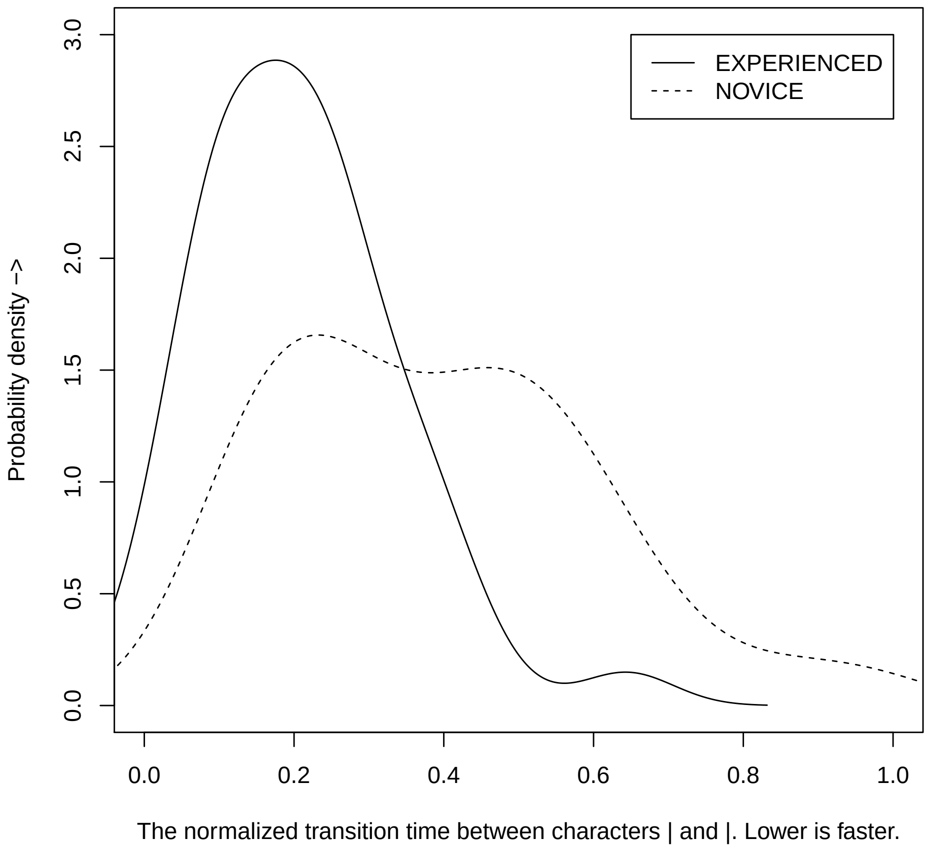

Figure 3: Smoothed probability density function of the times taken to press the character | twice by novice and experienced programmers.

dents' emotional states to engaged, neutral, and bored. They first measured emotions by filming students while they watched a video and had to self-evaluate their emotions at different points of the video. Then they observed students' reactions during the writing sessions and compared them to the emotions during the video session. They evaluated many machine learning methods and achieved up to 87% accuracy at classifying the student either bored or engaged based on the keystroke latencies. They note that results were not as good when instead of only binary classification into bored and engaged, the neutral emotional state was taken into account.

Vizer et al. [73] used keystroke features, pausing behavior and linguistic features for detecting stress. They designed a study where participants took part in four different experiments: two under no stress and two under stress.

The stress was induced by having participants complete cognitive or physical tasks before having them write a typing sample. The keystroke features they used included both relative amounts of different types of keystrokes, e.g. amount of arrow and space keystrokes as well as time per keystroke, i.e. the typing speed. The linguistic features they used included e.g. the relative amounts of nouns, verbs, word lengths, and positive and negative connotations of words. After the experiments, they used machine learning methods such as decision trees, support vector machines, and k-nearest-neighbor to classify typing samples into those written under stress and those written without stress. They were able to achieve accuracies of up to 75%. In addition to studying classification, they examined which features were best at classification. They found that for both physical and cognitive stress, relative amounts of some keystrokes were good at prediction. Interestingly, for physical stress, the average pause length had good predicting power, while for cognitive stress, the average time per keystroke had good predicting power, but neither feature had good predicting power for both physical and cognitive stress.

Compared to Bixler et al. [9] and Vizer et al. [73], Epp et al. [22] researched emotional states more generally. While Bixler et al. and Vizer et al. only studied boredom and engagement, and stress respectively, Epp et al. examined 15 emotional states. Epp et al. collected the data for their experiments by having a program run in the background while participants were on the computer. The program collected all keystrokes the users typed and would occasionally ask the user about their mood, i.e. emotional state. After filling the emotional state questionnaire, the user typed a fixed text. The 15 emotional states were measured by statements the user could agree or disagree with such as "I feel bored" or "I am focused". After collecting the data, Epp et al. extracted keystroke features such as digraphs and trigraphs out of the raw data. After feature selection, they used decision trees to classify typing samples based on emotional state and evaluated their model with 10-fold cross-validation. Their results indicate that some emotional states can be inferred from typing patterns better than others; for example tiredness and sadness were inferred very well, but relaxation and excitement were not. Classification accuracies of over 85% were achieved with some emotional states, e.g. tiredness and sadness.

2.5 Data Anonymization and De-Identification

In this subsection, we first describe identifiers that can be present in data, and then review a few studies on data anonymization focusing on the data anonymization methods in the studies.

2.5.1 Identifiers in Data

There can be different types of identifiers in data. *Explicit identifiers* are identifiers that can be used for identification by themselves [23]. For example, in medical records these could be names and social security numbers, while in source code snapshots these could be the student number and name of the student. They are easy to delete by going through the data and identifying certain attributes as explicit identifiers to be deleted. Deleting them is also sensible from the viewpoint that no valuable information is lost when they are deleted. For example, in keystroke latency -based studies, the student number or the name of the student are very unlikely to correlate with any of the other attributes in the data, and even if they do, it is almost certainly just due to randomness. The only case where there might be correlations that are not random are when some other variable is causing the correlation, which means that even in this case the explicit identifiers are conditionally independent with all the attributes in the data. For example, student numbers will probably correlate with the year the students started studying as student numbers are assigned in a sequential order. Furthermore, the starting year will probably correlate with the amount of credits a student has. In this case, the real correlation is between the starting year and the amount of credits, so given the starting year, the amount of credits and the student number will not correlate, i.e. it is not possible to predict the amount of credits based on the student number for all students who started studying in a certain year.

A harder challenge is to first identify *quasi-identifiers* [23] and then figure out what to do with them. Quasi-identifiers are identifiers that can not be used for identification in isolation, but can be combined together with other quasi-identifiers to identify individuals. For example, age is not an explicit identifier, since there are millions of people who all share the same age. However, if age is combined with other quasi-identifiers, accurate predictions about identity could be made. For example, it could be possible to identify who the person is if it is told that they are a 24-year-old research assistant living on a certain street in Helsinki and that they study computer science for the fifth year at the University of Helsinki. However, simply identifying attributes as quasi-identifiers is not enough. One could propose to delete such data altogether, but valuable information could be lost in the process [17]. For example, in keystroke data, the timings of the keystrokes are quasi-identifiers. A single timing will not reveal the identity of the typist, but combining all timings can be used to build a typing profile which can be used for identification [47]. Simply removing all keystroke timings would prevent identification based on typing patterns, but then the data could not be used for other keystroke latency -based studies such as inferring programming experience of the typists [42].

As removing all quasi-identifiers can reduce the value of the data significantly, quasi-identifiers are sometimes not removed from the data, but modified somehow to make it harder to use them for identification purposes. For example, in generalization [64], quasi-identifiers are modified to be more common: as an example, age could be reported to be within a certain range, e.g. 18-25, which would in turn make it harder to use it for identification. However, while modifying quasi-identifiers can make identification harder, it is not guaranteed to do so. For example, if there is only a single person in the data between the ages of 18 and 25, then modifying their age from the exact value to the range will not have an effect on identification, but will reduce the quality of the data. k-anonymity [67] is a measure for the degree of anonymity of data that has been developed to guarantee that anonymized, or de-identified, data can not be re-identified, i.e. that individuals can not be identified from the data. In k-anonymity, the goal is to make each individual's data similar to at least $k - 1$ other individuals; in other words, divide the data into blocks of k indistinguishable individuals who are essentially the same from the identification viewpoint. While k-anonymity can guarantee that individuals can not be identified, it might reduce the quality of the data. For example, in keystroke latency studies, forcing at least k individuals to have exactly the same typing profile would likely make inferring other information than identity infeasible as well. The following section will present some studies related to data anonymization in more detail.

2.5.2 Data Anonymization

Anonymity in data is often achieved by removing attributes from the data [23, 56, 66], reducing the accuracy of the data, e.g. by grouping and smoothing [29, 37] and by adding noise or fake information [21, 37]. Sun and Upadhyaya have developed a rule-based data sanitization method to remove sensitive information such as social security numbers from keystroke data [66].

Fung et al. outline four different types of attributes in data which reserve privacy: explicit identifiers, quasi-identifiers, sensitive attributes, and non-sensitive attributes [23]. As an example of anonymizing data by removing explicit identifiers and quasi-identifiers, network measurement data could be anonymized by removing attributes such as packet payloads and ip-addresses [56]. Daries et al. [17] analyzed the anonymization of data collected on MOOCs. They found two explicit identifiers – username and ip-address – and six quasi-identifiers – country, age, gender, and level of education of a participant as well as course id and the amount of forum posts – in their data and removed them.

In addition to removing attributes, other approaches for preserving anonymity have been suggested. For example, He et al. [29] suggested anonymization of set-valued data by distributing the data into buckets. Their work was motivated by the fact that the previously suggested approaches work well only if a subject is associated with a single sensitive value at a time, which does not suit set-valued data well. Similarly, Samarati et al. suggested replacing values in the data by semantically consistent less precise alternatives [64], i.e. generalization or rounding. A challenge here is to find an optimal degree of anonymization where data is minimally distorted while identification of subjects is still made improbable.

Recently, Monaco and Tappert developed two obfuscation strategies in the context of a third party continuously recording keystroke data [49]. They were able to decrease identification accuracy on average by 20% by adding a 25 ms random delay to the keystroke events and found that a delay of 500 ms was needed to reduce identification accuracy by half. In the context of a constant flow of keystrokes, there is a constraint that the anonymization should not affect the user experience, e.g. an added delay can not be noticeably long. However, in our context of open data sets there is no such constraint, which allows calculating optimal degrees of anonymization post hoc.

3 Research Design

In this section, we will present our research design for this work. We will formulate the research goals and questions and detail the data used in the experiments. We describe the context of the data, i.e. the courses from which the data is gathered and the data gathering mechanism.

3.1 Research Questions

In this work, we seek to determine how different degrees of anonymization of programming course data affects attributes that can be inferred from typing profiles. Our research questions are:

RQ 1. How does anonymization by rounding keystroke average latencies affect identification accuracy?

RQ 2. How does anonymization by bucketing affect identification accuracy?

RQ 3. How does anonymization affect inferring programming experience from typing profiles?

With the first research question, we seek to determine how rounding average latencies can be used to anonymize keystroke data. Rounding is similar to generalization [64] in that it makes the data less exact. We hypothesize that rounding the individual latencies in the data enough will affect the average keystroke latencies enough to render identification hard. Essentially, by rounding the latencies in the data, we hope that the typing profiles of the individuals will be adequately similar to each other so that the profiles, and thus individuals, can not be differentiated.

With the second research question, we explore whether splitting the data into even-sized buckets works for anonymization. Distributing the data into buckets, or bucketing, is in many ways similar to rounding the values. In both rounding and bucketing, the data is generalized, i.e. made less exact. We hypothesize that bucketing the keystroke latencies can be used for preventing keystroke latency -based identification similar to rounding the values.

Finally, with the third research question, we examine the extent of anonymization one can perform whilst still retaining information about programming experience. We are interested in finding an optimal amount of anonymization where identification is no longer practical, but programming experience can still be inferred. To study this, we will conduct experiments with varying amounts of anonymization, comparing identification accuracy with classification accuracy. We set a requirement that individuals should not be able to be identified from the anonymized data based on their keystroke latencies. We try to find a "sweet spot" where the programming experience of the individuals could be inferred but the identity could not. This would provide sanguine expectations that valuable information can be retained in anonymized keystroke latency data.

The overarching goal of this work is to seek an optimal anonymization procedure for keystroke latency data in order to remove a quasi-identifier from such data. Finding a procedure that could anonymize keystroke latency data while preserving valuable information would be a first step towards releasing such data openly to others. So far, the possibility of identifying individuals in such data has been a barrier to for example providing source code snapshot data sets openly for replication studies.

3.2 Context

The data used in the experiments comes from two similar introductory Java programming courses held in the autumns of 2014 and 2015 at University of Helsinki. Both courses lasted for 7 weeks. The courses taught the students programming basics such as variables, loops, input, and output. Both data sets were used in the identification experiments, but only the autumn 2014 course had information available on students' programming background, and therefore was the only one included in the programming experience experiments. These data sets are the same as the ones used in previous studies by Leinonen et al. [41,42], Longi et al. [47], and Leppänen et al. [43,44].

The students used an integrated development environment (IDE) for working on the course assignments. Using an IDE accustoms the students to professional software development tools from early on in their studies. The IDE recorded a snapshot for each action where the student modified the code while they were programming. The snapshots have a nanosecond level timestamp in addition to keystroke information. Students could turn the data gathering mechanism in the environment off if they chose to – data for this study was provided on a voluntary basis and no incentives were given to students who provided the data.

3.2.1 Programming Courses

The seven week long introductory course teaches students basic programming concepts. During the first week, students learn variables, printing output and reading input, and the while loop. On the following week, they are taught the usage and construction of methods. In the third week, the students get a taste of object-oriented programming and lists, and these topics are taught in more detail on the rest of the weeks. On the last week, students learn the basics of tables and sorting. The structure of the course was the same in both 2014 and 2015.

The main learning material on the courses is an online material, which includes text sections detailing programming concepts blended with assignments and questionnaires. Most students follow the material in a linear fashion, first reading about a concept, and then completing accompanying assignments and questionnaires related to that concept [1].

While there is a single two hour lecture each week on the courses, the focus is heavily on the weekly programming assignments. In the 2014 course, two thirds of the grade were based on the amount of assignments that were completed correctly, and in the 2015 course, exercise points accounted for 70% of the grade. The course assignments are accompanied by unit tests, which the students can run locally to check whether their program corresponds to what is required in the exercise. When the student is ready, they can submit their answer to the server, where unit tests are run again. Some of the harder exercises have so called "hidden tests", which are only run on the server, to prevent students from hard-coding their program to only pass the local tests.

There are about 10-20 assignments per week and the assignments for a single week increase in difficulty. The last couple of assignments of a week are often open-ended and more difficult than the previous assignments. They tie together many of the concepts learned on the week and correspondingly yield more points. The students can get help on the assignments in lab sessions, where the extreme apprenticeship [40, 70] method is used. Extreme apprenticeship means that the teaching assistants in the lab sessions are mostly students who have only completed the course very recently, usually in the previous year. Lab sessions are organized almost daily, and there is around 20 hours of lab sessions per week.

3.2.2 Data Gathering

Most of the students use the integrated development environment (IDE) NetBeans with a custom plugin called TestMyCode (TMC) [57] when working on course assignments. TestMyCode is used by the students to download the weekly exercises, test whether their solutions pass the unit tests that come with the exercises, and submit their correct solutions to the server to get credit for completing the exercises. TestMyCode has an option to collect source code snapshots for research purposes: the students are allowed to turn off data collection and do not receive any incentives such as course points for providing their data. TestMyCode only collects data from the course exercises, notably, it does not collect data from personal or work-related programming projects. Figure 4 illustrates how TMC works.

Source code snapshots are collected after any change is detected in the text content, i.e. the source code, and when the students run unit tests or submit their code to the server for evaluation. A limitation of this is that

[1]Confirmed in yet unpublished research.

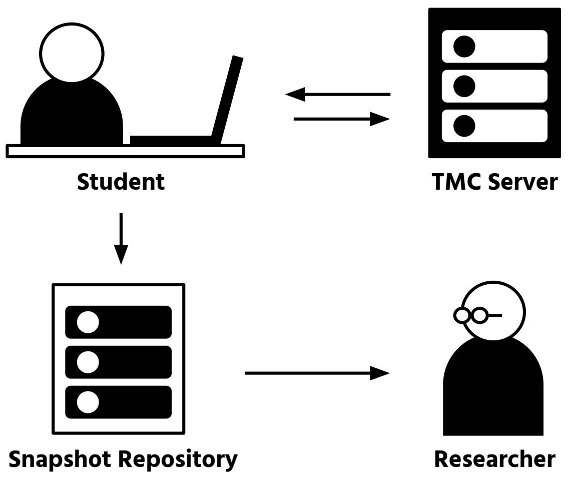

Figure 4: Source code snapshots are collected by the TestMyCode (TMC) plugin in the NetBeans IDE. Students can send their solutions to the TMC server, which gives students credit for correct solutions. If the students do not opt out of data gathering, TMC will also continuously collect source code snapshot data and send them to a snapshot repository for research purposes.

only visible source code changes are recorded as keystrokes, for example, pushing of the control- or the shift-key is not recorded. In addition to the visible changes, the source code snapshots also include a timestamp, student id, course id, and assignment id. Latencies between keystrokes can be derived by observing changes between subsequent snapshots – for example, if the last character of a snapshot is a, and the next snapshot has the character b added after a, and the timestamps of the snapshots are c and d, then we can deduce that the observed latency of the digraph ab in this case is $d - c$ time units. In our source code snapshots, the timestamps are in nanoseconds, which are converted to milliseconds in the analysis in this work.

It should be noted that from the snapshots alone, we do not know which computer and keyboard the students are using during an exercise. Students can take advantage of the computer labs at the University where teaching assistants can provide help on the exercises. However, the students can also complete the exercises individually in any environment they find most suitable. Due to this, students can even change computers (and keyboards) within a single exercise.

For the programming experience inference experiments, we used data from a background survey the students can answer in the beginning of the course. The background survey has mostly questions about demographics such as age and gender, but also about programming experience. For the programming experience inference experiments, we categorized the students into two cohorts – those with no previous programming experience and those with at least some programming experience.

4 Methodology

In this section, we outline our research methodology for conducting the experiments and answering our research questions.

4.1 Data Preprocessing

For preprocessing the keystroke data, we followed the procedure outlined in the study by Longi et al. [47]. Only digraphs with latencies between 10 ms and 750 ms were included as first done by Dowland and Furnell [20]. The lower bound is necessary to eliminate auto-completion events from the IDE or cases of two keys being struck together accidentally. The upper bound is needed to only capture the subconscious typing rhythm of the student and to remove any breaks they might take while working on an exercise.

Since the typing profiles are built with average latencies, we required that a student should have at least five occurrences of any digraph used to build their typing profile as first done by Killourhy and Maxion [38]. If the student had only typed a digraph under five times, the average latency for that digraph was excluded from the student's typing profile. Snapshots where multiple characters were added at the same time were discarded as they were almost exclusively copy-paste events.

The students who did not volunteer to provide their programming background details were excluded from the programming experience study. The autumn 2014 course had 199 students, of which 82 students (41.2%) had at least some programming experience and 117 (58.8%) had none. The autumn 2015 course used in addition to the 2014 course in the identification experiments had 153 students. We excluded students who opted out of the data gathering, as well as those who typed less than 2000 characters during the first week of the course. On average, the students type 7500 characters during the first week, which means that only students who worked on more than one quarter of the first week were included. This was done so that students who drop out or turn off the data gathering mechanism on the first week do not introduce unnecessary noise to the data analysis. After preprocessing, there were 199 students left in the autumn 2014 data set and 153 in the autumn 2015 data set.

4.2 Identification

For the identification experiments, we use the acceptance threshold method introduced by Longi et al. [47] where a match in the top k closest training set samples is considered correct for a specific test sample. The idea behind this is that exact identification is not always mandatory. For example, for authentication in online exams, it is sufficient to be quite sure that the students are who they claim to be. The parameter k controls the balance between two types of error: larger values of k increase the identification accuracy, but also make it easier for impostors or cheaters to masquerade as the genuine user. The probability for an impostor being successful is k/n, where n is the number of students, assuming the typing profiles are uniformly distributed, i.e. random. In a real world scenario, an impostor could possibly have at least partial information about the typing profile of their victim or the true distribution of the typing profiles, and would therefore be able to increase their chances of impersonating the victim. Optimally, the smallest value of k that has sufficiently high identification accuracy should be used.

To build the typing profiles, the average latency between two specific characters was calculated for all character pairs, i.e. digraphs, for each student in the data. In addition to the average digraph latencies, the average typing speed of the user was included in the typing profile. If a student had not typed a digraph, the missing value was replaced with the student's average typing speed. After this, the different anonymization procedures were performed with the data. The anonymization process is analyzed in more detail in Sections 4.5 and 4.6. After the anonymization procedure, the typing profiles still consisted of average digraph latencies and the average typing speed in milliseconds. Before calculating the distances, the values were normalized to be within the range from zero to one. This was done so that all digraphs would affect the distance equally, i.e. that digraphs with large average latencies would not affect the results more than digraphs with smaller average latencies. For example, digraphs between special keys and keys that are far apart from each other on the keyboard will understandably have large average latencies, while keys that are near each other will have small average latencies. We consider that the relative differences between digraph latencies of subjects are more telling of the identity of the subject compared to absolute differences. Normalization was done with Equation 1, which modifies each feature $x_i \in \mathbb{R}_{\geq 0}$ into $x_i' \in [0, 1]$. x_{min} denotes the minimum value of the feature in the data and x_{max} denotes the maximum value of the feature in the data. For example, if the minimum value for the feature in the data was 100 milliseconds and the maximum value was 500 milliseconds, a value of 300 milliseconds would be converted to be 0.5, since it is in the middle of the range from 100 milliseconds to 500 milliseconds. The resulting feature vectors for the students were vectors of values between 0 and 1.

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{1}$$

For both data sets used in the identification experiments, we chose to build the typing profiles in the training set from the first six weeks of exercises and used the data from exercises of the last week as the test set. To determine if a test sample was correctly identified, we calculated the euclidean distance to each training set sample. Euclidean distance is calculated with Equation 2, where $\bar{x}$ is the feature vector for one student and $\bar{y}$ for the other, and n is the amount of features. The feature vectors include the average digraph latencies. In the cases where one of the students did not have a value for a specific digraph, i.e. x_i or y_i in Equation 2, their average typing speed was used instead. After having calculated the distances from the test sample to the training samples, the training samples were sorted based on the euclidean distance from the test sample. We used an acceptance threshold of $k = 10$, and thus regarded the student to be correctly identified if their typing profile was in the top 10 closest training set matches.

$$d(\bar{x}, \bar{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad (2)$$

4.3 Programming Experience Inference

Earlier research indicates that the Bayesian Network and Random Forest classifiers have good performance at classifying students in the context of inferring programming experience from typing profiles [42]. Therefore, we classify the students into two groups: those with some programming experience and those with none using the Bayesian Network, Random Forest, and majority classifiers. We chose to use 100 decision trees in the Random Forest with each tree considering six random features. This was done so that the decision trees within the Random Forest would not correlate and thus overfit their training data [10]. The majority classifier will classify every sample to the majority class, and is therefore good as a baseline against which the performance of the other two classifiers can be measured. The classification accuracy is evaluated using 10-fold cross-validation.

4.4 Feature Selection

In this subsection, we first review what kind of an effect feature selection can have on identification accuracy. Then, we present the features that will be used in the experiments, motivations for selecting those features, the issue of overfitting that feature selection in general can both cause and cure, and the measures we take to avoid overfitting in our experiments.

4.4.1 Effect of Feature Selection on Identification Accuracy

Leinonen et al. [41] have studied the effect of feature quantity on identification accuracy to see how the feature count influences the identification accuracy. Their results show that for identifying students precisely, i.e. with an acceptance threshold of one, 50 features seem to be enough as the increase in accuracy with 100 features is not significant enough to warrant increased complexity. If a student is allowed to be within an acceptance threshold of

Table 1: The 25 most common digraphs in the 2015 introductory course data set which were used in building the typing profiles for that course [41].

from key	space	t	n	l	a	a	=	r	j	t	{	l	h
to key	=	u	t	u	r	t	space	i	a	h	}	i	i

from key	i	s	o	backspace	k	i	t	v	u	t	e	u
to key	n	t	u	backspace	u	s	a	a	k	space	t	t

5 or 10, 25 features seem to suffice. Their results for the 2015 introductory course data set are presented in Figure 5. That same data set will be used in our identification experiments later in this work. The figures for other data sets showed similar diminishing returns in the increase of accuracy after around 25 features. The 25 most common digraphs for the same set are presented in Table 1. Since the digraphs are from a programming context, the most common ones include digraphs related to programming such as the digraph used in creation of Java's code blocks *{ -> }*, and *i -> n* and *n -> t* from writing *int* which defines the type of a variable to be integral.

Their conclusion is that 25 digraphs is optimal as adding more features beyond 25 seems to only marginally increase identification accuracy. However, adding more could possible result in overfitting, which is to be avoided.

4.4.2 Identification

In the study by Longi et al. [47] that introduced the acceptance threshold method we use in our identification experiments, the typing profiles were constructed using three different types of features: 1. the average latency between any two keys, i.e. the typing speed of the student, 2. single character latencies, i.e. the average latency from any key to a specific key, and 3. digraph latencies, i.e. the average latency from a specific key to a specific key. Since digraph latencies have been shown to work better than the other two types or a combined feature vector that includes all three types [47], we only use digraph latencies in our experiments.

For exploring how identification accuracy suffers when the data is anonymized (RQ1 & RQ2), the 25 most common digraphs were used to construct the typing profiles as first done by Leinonen et al. [41]. The most common digraphs were determined by sorting the digraphs by the median amount of times the students had used them in both the training and the test set. One digraph corresponds to one feature, and for a feature to be included from a specific student, the student had to have at least five instances of the digraph in the data.

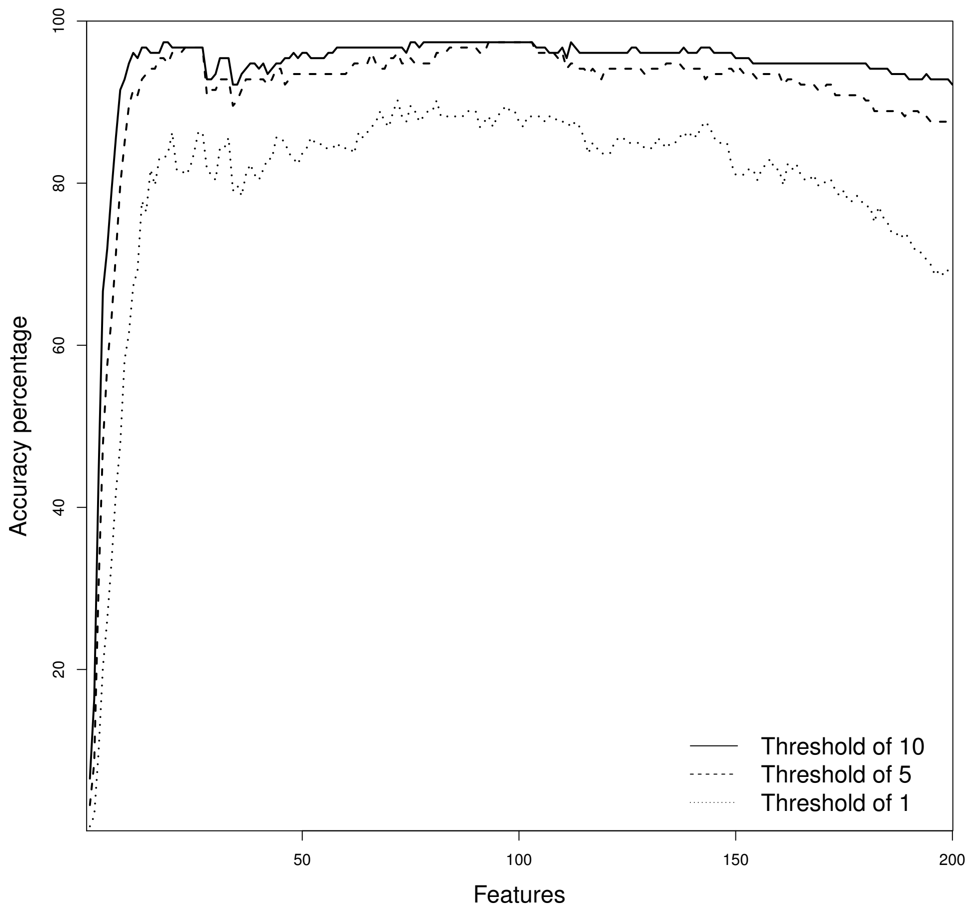

Figure 5: Smoothed identification accuracy plotted against the number of features. The threshold specifies the number of students that are considered to be correct for identification purposes. The figure shows that using around 25 features provides a good identification accuracy with all three thresholds and that the accuracy starts to deteriorate after around 150 features, especially with a threshold of 1, i.e. exact identification [41].

4.4.3 Programming Experience Inference

In the feature selection for the programming experience data set (RQ3), we followed the procedure outlined by Leinonen et al. [42] for inferring programming experience from typing profiles. Features with no data (e.g. digraphs that no student had typed) were removed. Then, the BestFirst feature selection algorithm in the WEKA Data Mining toolkit [28] was used for feature selection. The BestFirst algorithm starts with an empty set of features, populating it one by one and observing the effect on identification accuracy. If identification accuracy does not improve for five additions in a row, the algorithm backtracks and removes features that did not increase identification accuracy. In other words, it greedily searches the parameter space for optimal features to include in the feature set. Out of more than 10000 initial features, less than 50 features were included in both the 2014 and the 2015 data sets after the feature selection. Feature selection can reduce overfitting, shorten training times, and improve the interpretability of the resulting model, i.e. make it more understandable. For example, for inferring programming experience, it would be understandable that the average latencies of programming related digraphs would best predict the programming experience of the student, since expert programmers probably type those digraphs faster than novices on average. Thus, we would expect our model to be based on those digraphs.

4.4.4 Avoiding Overfitting

A concern that arises from feature selection is the possibility of overfitting the model to the training data. Overfitting happens when the model does not generalize well due to being too complex. While an overfitted model might achieve good results on the training data, it might be poor with new data that the model was not exposed to during training. A model can become too complex for many reasons, for example due to adding unnecessarily many parameters to the model. While feature selection is usually used to reduce the amount of features in the model, for example in our experiments, and consequently reduce overfitting, the choices made in selecting the features can result in overfitting if the selected features do not generalize to other data sets. Some features might be good at prediction overall within a domain, while others just happen to be good with a certain data set. For example, the average latency for a programming related digraph will probably generalize quite well and be good at predicting programming experience regardless of the data set. However, since the amount of features can be huge (e.g. over 10000 in our experiments), there will likely exist features that are not programming related, but still have good predictive power for programming experience in some data set.

As a somewhat extreme example of overfitting in the context of identifying students, consider a scenario where we are trying to predict students correctly in a data set with 200 students. Let's say we have black box model, i.e. we have no idea of how the model works internally, but we know that it uses 200 digraph average latencies as features. What we would optimally want in this scenario is that the students are identified based on their average digraph latencies. However, what could happen in theory, assuming we do not know how exactly the model is fitted, is that the model learns to identify the students so that one feature corresponds to each student.

For example, it could learn that student A has an average latency of 500 milliseconds for the digraph *ab* in the training set, and only ever use the digraph *ab* when identifying student A. The model would achieve perfect identification accuracy on the training set, but would not be able to identify the students in a test set very well, since it is highly unlikely that the students would have the exactly same average latency in the test set at the millisecond level. Another possibility could be that the model only learns to identify students based on unique digraphs the student may have in the training set. For example, for each student, it could find a digraph that only the examined student has typed, and identify based on the existence of that digraph in the typing profile. Again, the model would achieve a perfect 100% identification accuracy with the training set, but it is not guaranteed that the unique digraphs are the same in the test set. Thus, it would again most likely perform poorly with the test set.

We try to limit the possibility of overfitting in multiple ways. First of all, in the identification experiments, we only use the 25 most common digraphs. This means that there are a lot less features than students, which reduces the chances of overfitting. In the programming experience inference experiments, we use the WEKA Data Mining toolkit [28] for feature selection. Less than 50 features out of the initial over 10000 were left in the data set afterwards. The features were selected based on their predictive power over programming experience, i.e. we used the features that are best at predicting the value of the programming experience variable. The features, in this case digraphs, were confirmed by the author to be sensible – most were obviously programming related such as the OR-clause in conditions *//* and *i+* from typing *i++* which is used in Java to increment an index variable. Note that these are not necessarily the most common digraphs opposed to the identification experiments in which the 25 most common digraphs were used.

4.5 Anonymization by Rounding

We use an anonymization technique similar to generalization [64] where the values in the data are rounded to reduce identification accuracy (RQ1). To investigate how rounding the average latencies in typing profiles affects programmer identification and classification based on programming experience, we modified the latencies using Equation 3. It rounds the latency z to the nearest x, where x is the number of milliseconds given to the anonymization function as a parameter. The resulting value y is then used instead of the original value z in the construction of the typing profile. The aim is to reduce the accuracy of the data, hopefully reducing identification accuracy in the process, which would anonymize the data. We studied how identification accuracy deteriorates when the value of x is increased.

$$y = x * round(z/x) \tag{3}$$

Equation 3 essentially distributes the average latencies into buckets. For example, if x is 100 milliseconds, all latencies will be rounded to the nearest multiple of 100. This leads to all latencies between 0 and 50 ms being rounded to 0 and distributed to the first bucket, all latencies between 50 and 150 ms being rounded to 100 and distributed to the second bucket, all latencies between 150 and 250 ms being rounded to 200, and so on.

After rounding the average latencies, the data was normalized to reduce the effect of digraphs with large average latencies on the distance calculations. Then, the euclidean distances from each test sample, i.e. the typing profile from last week of the course, to the training samples, i.e. the typing profile from the first six weeks of the course, were calculated. For each test sample, the closest ten training set samples were checked ($k = 10$ in Longi et al. [47]'s identification model), and if any of them were authored by the author of the test sample, the author of the test sample was considered correctly identified.

4.6 Anonymization by Bucketing

The buckets that result from the rounding method are not equal in size: the size of the first bucket is half the size of the subsequent buckets. Motivated by this we analyzed whether distributing data into even-sized buckets could be used for anonymizing keystroke data (RQ2). We modified the average latencies in the data by first increasing each latency z by half of the size b of the buckets using Equation 4, and then rounding each latency z_1 to the nearest x, where x is the current bucket size b using Equation 5. The resulting value y is then used instead of the original value z in the construction of the typing profile.

$$z_1 = z + (b/2) \tag{4}$$

$$y = b * round(z_1/b) \tag{5}$$

The only difference between this method and the rounding method is that this method distributes the data into even-sized buckets. For example, if we have buckets of 100 milliseconds, we want all latencies between 0 and 100 milliseconds to be in the same bucket. Now, any latency between 0 and 100 ms will first be incremented by 50 ms (half the bucket size), leading to a distribution between 50 and 150 ms. Then, the latencies will be rounded to the nearest multiple of 100 milliseconds (the bucket size), which in the case of values between 50 and 150 milliseconds is 100 milliseconds. The procedure is then repeated for all values between 100 and 200 milliseconds, etc.

Again, after rounding the average latencies, the data was normalized, the distances between test and training samples were calculated, and if the ten closest training samples as measured by euclidean distance contained the training sample of the author of the test sample, the test sample's author was considered correctly identified.

Table 2: Identification accuracy percentages with different rounding precisions.

	Original	100 ms	200 ms	300 ms	400 ms	500 ms	600 ms
2014 course	98.0	81.7	6.5	72.5	77.1	6.5	6.5
2015 course	97.8	81.3	56.8	67.6	67.6	7.2	7.2

5 Experiments and Results

In this section, we describe the experiments we conducted to answer each of the research questions and the results of the experiments. We first analyze how anonymizing data with the rounding method described in Section 4.5 and the bucketing method described in Section 4.6 affect identification accuracy with increasing amounts of anonymization. Then, we examine how the methods affect inferring programming experience from the data.

5.1 Identification with Anonymized Data

For the identification experiments, we use the acceptance threshold method introduced by Longi et al. [47] where a match in the top k closest training set samples is considered correct for a specific test sample. For example, with an acceptance threshold of ten, identification is deemed to be correct if the ten closest typing profile matches from the training set include the typing profile of the test sample student. The distance between two typing profiles was calculated by the euclidean distance between the features, i.e. average latencies.

We had two data sets for the identification experiments. For both data sets, we chose to build the typing profiles in the training set from the first six weeks of exercises and used the data from exercises of the last week to build the test sets. To determine if a test sample was correctly identified, we calculated the distance to each training set sample. We then sorted the training set samples based on the distance from the test sample. We used an acceptance threshold of 10, and thus regarded the student to be correctly identified if her typing profile was in the top 10 closest training set matches.

5.1.1 Identification with Rounding

To answer the first research question, *"How does anonymization by rounding keystroke average latencies affect identification accuracy?"*, we calculated identification accuracies with different amounts of anonymization using the rounding method.

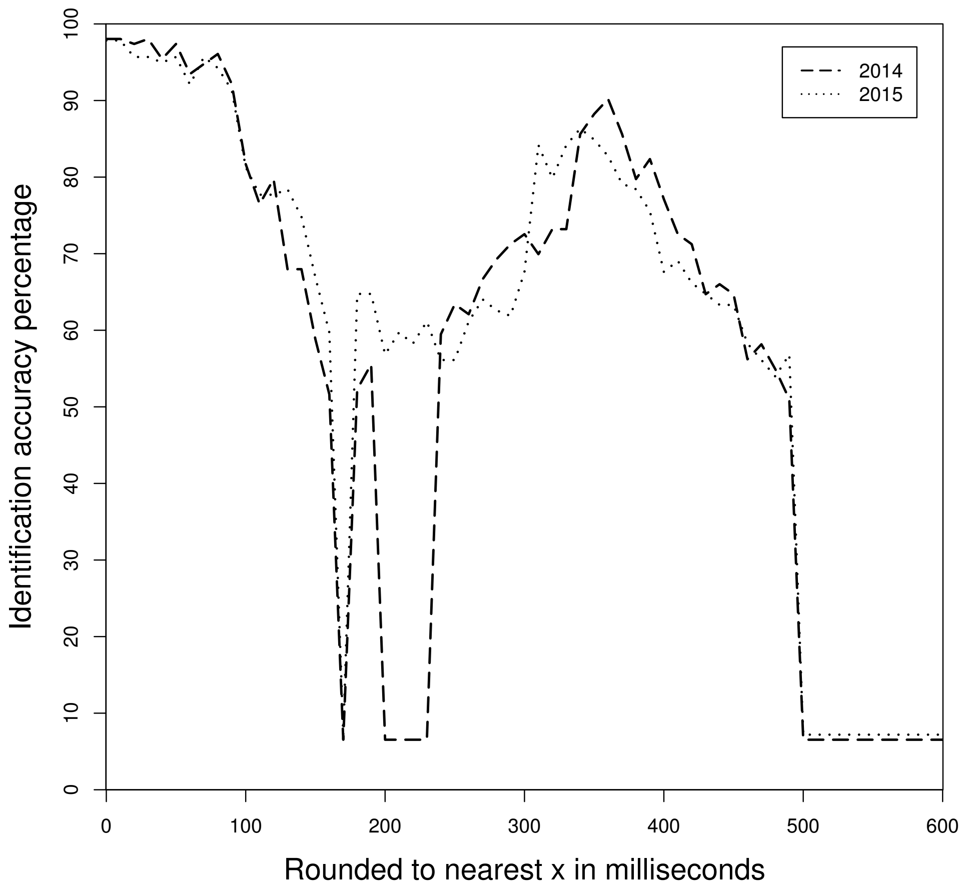

Figure 6: Identification accuracy compared against rounding precision. All values in the data were rounded to a nearest millisecond value. The larger the millisecond value in the x-axis, the lesser the rounding precision. The y-axis expresses identification accuracy.

The results of the experiments using the rounding method for anonymization are presented in Table 2 and plotted in greater detail in Figure 6. The millisecond values in the column header of the table represent the rounding, i.e. how much the latencies were rounded.

When using rounding for anonymization, identification accuracies in both data sets deteriorate in the first two 100 ms steps, but then get better or stay equal in the next two steps. After that they start declining again. The unexpected value of 6.5% in the rounding experiment of the 2014 data set when rounding to 200 ms, and the dip in accuracy between 150 and 350 milliseconds visible in Figure 6 is discussed in further detail in Section 6.2.

Table 3: Identification accuracy percentages with different bucket sizes.

	Original	100 ms	200 ms	300 ms	400 ms	500 ms	600 ms
2014 course	98.0	26.1	12.4	6.5	6.5	6.5	6.5
2015 course	97.8	31.7	15.8	7.2	7.2	7.2	7.2

5.1.2 Identification with Bucketing

To answer the second research question, *"How does anonymization by bucketing affect identification accuracy?"*, we calculated identification accuracies with different amounts of anonymization using the bucket method.

The results of the experiments using the bucket method for anonymization are presented in Table 3 and plotted in greater detail in Figure 7 The millisecond values in the column header of the table represent the bucket size.

Using buckets for anonymization, identification accuracies in both data sets deteriorate with all 100 ms steps and reach their lowest values already after three steps. These results are different from the results of the rounding anonymization method, where the lowest values were only attained after 5 steps and at the third step mark the identification accuracies were still quite high at around 70% accuracy compared to the around 7% accuracy with the bucket approach.

5.2 Programming Experience Inference with Anonymized Data

To answer the third research question, *"How does anonymization affect inferring programming experience from typing profiles?"*, we measured classification accuracies with different amounts of anonymization using both the rounding method and the bucket method and multiple classifiers.

Table 4 shows the classification accuracy results with different amounts of anonymization. With the rounding method, classification accuracies deteriorate slightly with each step, although there are exceptions. We do not observe a similar effect as with identification, where the accuracy temporarily improved when transitioning from rounding to nearest 200 milliseconds to rounding to nearest 300 milliseconds. Similar to the rounding method, classification accuracies with the bucket method degrade with each step. A clear difference is that with the bucket method the classification accuracies decline faster, nearing the performance of the baseline majority classifier when the bucket size is 600 ms. In contrast, with the rounding method, Bayesian Network and Random Forest outperform the majority classifier by over 10 percentage points at the 600 millisecond mark.

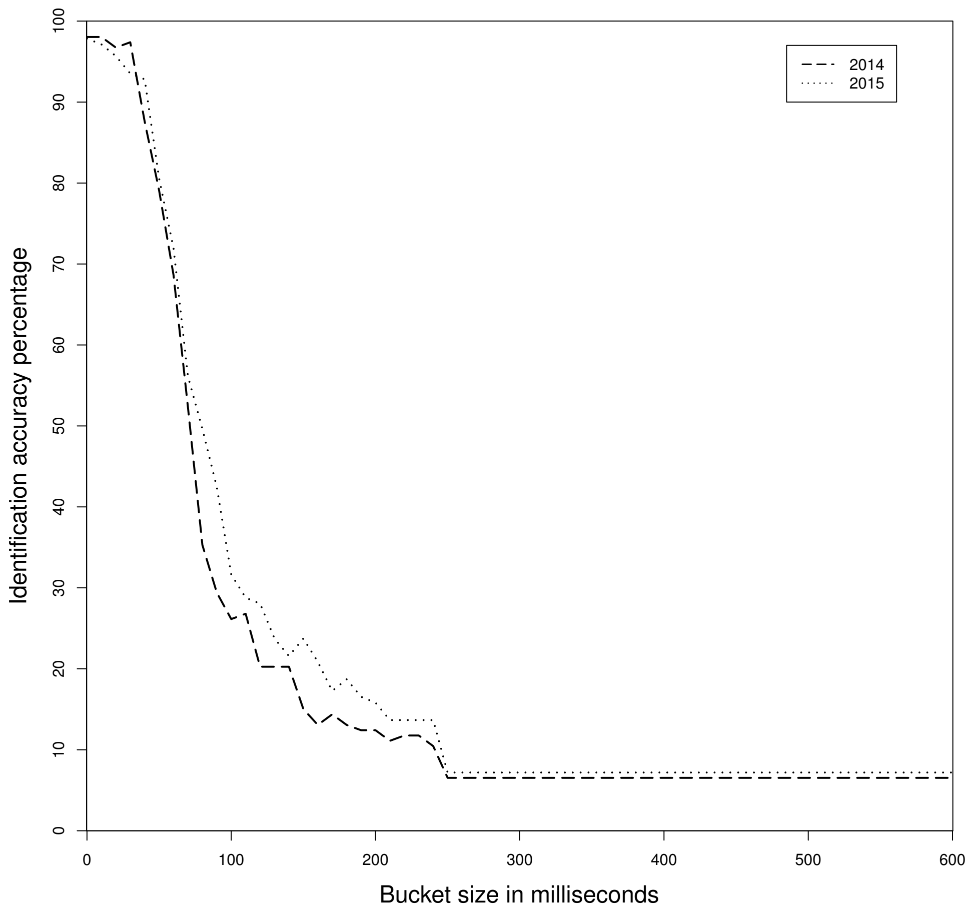

Figure 7: Identification accuracy compared against increasing bucket size. The data was split into even-sized buckets. The larger the millisecond value in the x-axis, the lesser the rounding precision. The x-axis represents bucket size and the y-axis expresses identification accuracy.

5.3 Identification vs Classification Accuracy

With both the rounding and the bucketing methods, programming experience classification accuracy does not decline as fast as identification accuracy. With the rounding method, you could choose to round values to 500 milliseconds, resulting in identification accuracy of around 7%, but a programming experience classification accuracy of 73.9% with both the Bayes Net and the Random Forest classifier, around 15 percentage points higher than the majority classifier.

With the bucketing method, a lesser degree of anonymization is needed,

Table 4: Programming experience classification accuracy percentages with different rounding precisions and bucket sizes.

Method	**Rounding**		
Classifier	Bayes Net	Random Forest	Majority Classifier
0 ms	75.4	73.9	58.8
100 ms	73.9	75.4	58.8
200 ms	73.9	72.4	58.8
300 ms	73.9	70.9	58.8
400 ms	68.3	73.4	58.8
500 ms	73.9	73.9	58.8
600 ms	70.4	71.4	58.8
Method	**Buckets**		
Classifier	Bayes Net	Random Forest	Majority Classifier
0 ms	75.4	73.9	58.8
100 ms	73.4	73.4	58.8
200 ms	71.4	75.4	58.8
300 ms	70.4	71.4	58.8
400 ms	61.3	69.8	58.8
500 ms	64.3	67.3	58.8
600 ms	58.8	60.3	58.8

since already with 300 millisecond buckets, reliable identification is no longer possible (accuracy is around 7%), but programming experience classification accuracy is still significantly higher (around 71%) than with the majority classifier.

The decline in identification and classification accuracy with the rounding method is shown in Figure 8 and with the bucketing method in Figure 9.

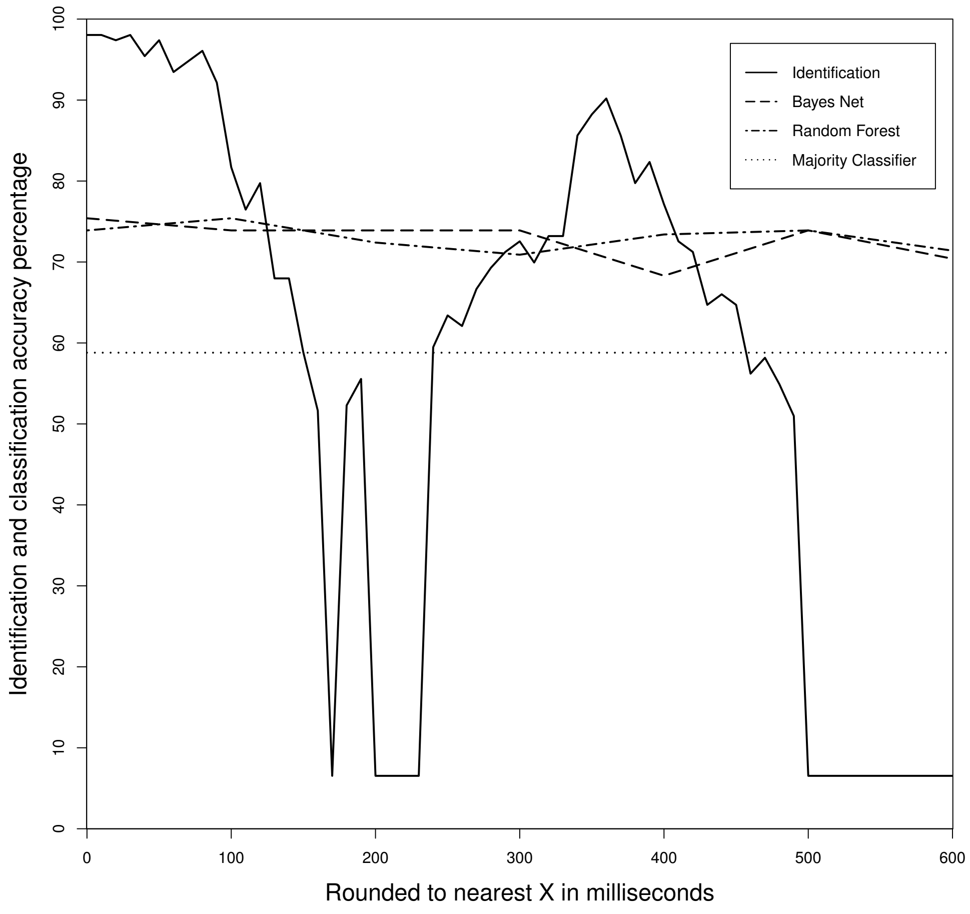

Figure 8: Identification (solid line) and programming experience (dashed lines) classification accuracy compared against increasing rounding amount. The data was rounded to nearest multiple of X. Programming experience classification accuracies are shown for three different classifiers: Bayesian Network, Random Forest, and the majority classifier. The x-axis represents rounding amount and the y-axis expresses identification and classification accuracy.

6 Discussion

In this section, we discuss our results. We first ponder the consequences of our results and their impact. Then, we analyze our results more deeply in respect of the amount of buckets instead of amount of anonymization. Even though they do correlate, the relatively small amount of buckets needed for quite reliable identification warrants further consideration. Lastly, we

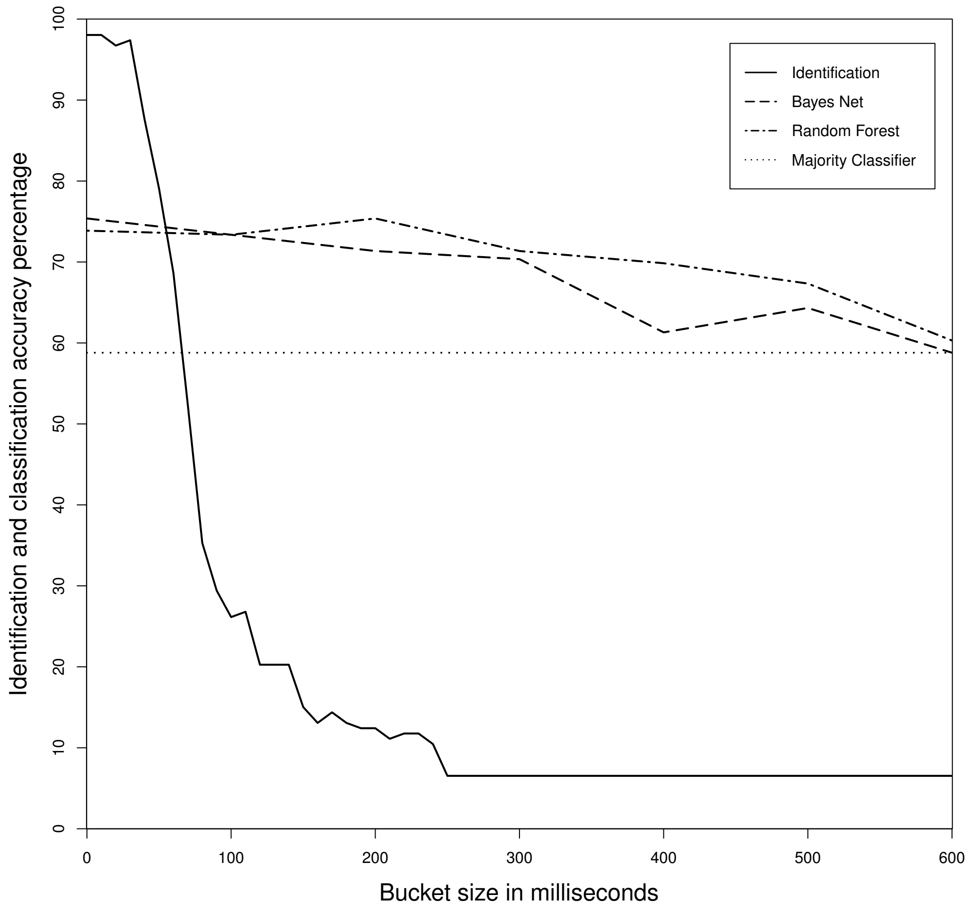

Figure 9: Identification (solid line) and programming experience (dashed lines) classification accuracy compared against increasing bucket size. The data was split into even-sized buckets. Programming experience classification accuracies are shown for three different classifiers: Bayesian Network, Random Forest, and the majority classifier. The x-axis represents bucket size and the y-axis expresses identification and classification accuracy.

discuss some limitations of our work.

6.1 Consequences of Results

In this work, we studied how typing profile data could be anonymized whilst retaining information important to researchers in the data. The motivation and long-term goal of the study is to be able to release open data sets where data that could be used to identify subjects is removed. We explored two different ways of anonymizing data consisting of student typing profiles on programming courses.

For the rounding method, rounding keystroke average latencies to the nearest 500 milliseconds would be optimal. When rounding to the nearest 500 milliseconds, reliable identification is not possible, since only around 7% of students are correctly identified with a threshold of $k = 10$, i.e. regarding the student correctly identified if their typing profile is in the top 10 closest matches, compared to the non-anonymized accuracy of around 98.5%. Nevertheless, with the same 500 millisecond rounding, programming experience can be inferred accurately for 73.9% students. With the Random Forest classifier, programming experience classification accuracy has remained the same as without anonymization, and with the Bayesian Network classifier, it declined only by 1.5 percentage points.

For the bucket method, the optimal amount of anonymization is quite different from the rounding method. With even-sized 300 millisecond buckets, identification accuracy has decreased to the lowest value it will reach. At that point, programming experience classification is possible with around 71% accuracy compared to the 58.8% accuracy with the majority classifier. The result indicates that the bucket method is more efficient at anonymizing the data, although more domain-relevant information is lost in the process.

The results demonstrate that at least in the context of inferring programming experience, keystroke latency data can be modified in a way that prevents keystroke latency -based identification, more specifically, identification with the model proposed by Longi et al. [47]. Namely, we were able to remove a quasi-identifier from the data in this context. The context of the results is important and should be considered when evaluating the results as the methods studied in this work have only been shown to be able to prevent a specific identification method, which does not necessarily guarantee that other identification methods would be affected. For example, identification based on linguistic features (see e.g. [14]) could very well still be possible as the textual content of the programming snapshots, i.e. the source code itself, was not modified. However, if further studies indicate that the anonymization methods introduced in this work can anonymize data in respect of other keystroke latency -based identification models as well, anonymized keystroke latency data could possibly be made open for researchers. Having open data sets would enable replication studies conducted with the anonymized data and also novel studies with existing data. For example, if open keystroke latency data was more abundant, this study and our previous keystroke -based studies [41–44, 47] would have likely included studies with data from other contexts in addition to our own in order to examine the generalizability of the results.

The anonymization methods suggested here do not guarantee any form of k-anonymity [67]. In other words, it is not guaranteed that k individuals in the data would be indistinguishable, i.e. have exactly same typing profiles. In our context, k-anonymity is likely too strict and would decrease the quality of the data significantly similar to what happened in Daries et al.'s Massive Open Online Course (MOOC) data de-identification studies [17] where 5-anonymity was required and results of studies on anonymized data were significantly poorer compared to studies on the original data. Already with the amount of anonymization we conduct, the amount of different buckets or options for the keystroke latencies is actually very small. This is due to the fact that only latencies between 10 and 750 milliseconds were included in the data. Thus, when the bucket size is for example 250 milliseconds, the keystroke latency values can only belong to either the 0–250, the 250–500, or the 500–750 millisecond buckets, i.e. only three values are possible. Still, since the amount of features is large, k-anonymity would probably force values to be even more similar in order to achieve k-anonymity, i.e. have $k - 1$ indistinguishable individuals for each individual in the data. Our hypothesis of k-anonymity being too strict is supported by our results, which indicate that keystroke latency -based identification can be prevented without achieving k-anonymity at least in our context. In any case, research conducted on anonymized keystroke latency data should consider whether the anonymization procedures could have affected the results similar to earlier results in social science research [17]. Yet another concern is whether other possibly existing data sets could be connected to the openly released data set as feared by Longi et al. [47] in the keystroke latency field, and as has happened with movie recommendation data [55].

6.2 Amount of Buckets

The results of the rounding method are interesting due to the fact that only keystroke latencies between 10 and 750 milliseconds were included in the typing profiles. When rounding to the nearest 500 milliseconds, there are only two possible values for the features – 0 milliseconds or 500 milliseconds – since all values between 0 and 250 milliseconds will be rounded to 0 milliseconds while values between 250 milliseconds and 750 milliseconds (the upper bound) will be rounded to 500 milliseconds. The result means that for inferring programming experience from typing profiles with moderately high accuracy, it is sufficient to categorize all average latencies that the typing profiles include into two buckets based on whether the student is fast or slow at writing the digraph.

Another interesting find is that when the rounding method is used, identification seems quite reliable with an accuracy of around 74% even when rounding to the nearest 300 or 400 milliseconds, while the results for rounding to around 100–200 milliseconds were significantly worse. To further examine this, we plotted the changes in identification accuracies in 10 millisecond intervals. The resulting plot is in Figure 6. The local maxima for the two courses are at 340 ms with 86.3% accuracy and 360 ms with 90.2%

accuracy. When rounding to both 340 and 360 milliseconds, there are only three buckets in our data due to filtering out events that are not between 10 and 750 ms. For example, with 340 milliseconds, values between 0 and 170 ms are rounded to 0 ms, values between 170 and 510 ms are rounded to 340 ms, and values between 510 ms and 850 ms are rounded to 680 ms. The local minima for both courses are at 170 milliseconds. At that point, there are five possible values, i.e. buckets for the data: 0–85, 85–255, 255–425, 425–595, and 595–765 millisecond buckets. This essentially means that using three buckets yields better results than using five buckets. More generally, the effect seen in Figure 6 implies that categorizing data into 3 buckets works better for identification than categorizing data into more buckets, unless the rounding starts to be insignificant (under 100 milliseconds).

Our hypothesis and a potential explanation is that additional buckets beyond three add unnecessary noise to the data. For example, with five buckets – very slow, slow, mediocre, fast, very fast – there might not be enough average latencies in the very slow and very fast buckets. On the other hand, some average latencies that should be categorized to the mediocre bucket for maximal performance might be categorized to the slow or fast buckets. Another possibility is that the average latencies are distributed unevenly so that many of them are assigned to the very first bucket in the bucket method, but into two different buckets with the rounding method. For example, consider a case where most of the latencies are between 0 and 200 milliseconds and we are rounding to 200 milliseconds. In this case, the rounding method would have one bucket with the values between 0 and 100 ms, and one bucket with the values between 100 and 300 ms, and so on. Thus, the rounding method would be able to differentiate the 0–100 ms latencies from the 100–200 ms latencies, but the bucket method would distribute all of them into the same bucket and could not differentiate between them. However, further research is needed to confirm or deny either of these hypotheses.

Only data from two introductory programming courses with very similar content were used in the experiments where the peculiar effect in Figure 6 was first observed. To investigate the effect further, we used data from an advanced programming course to see whether a similar effect of identification accuracy temporarily decreasing and then increasing occurs. Again, we plotted the changes in identification accuracy in 10 millisecond intervals. The plot for the advanced course is in Figure 10. Clearly, the effect is present also in the advanced course, where content is different from the introductory course: in the advanced course, the students learn more abstract programming concepts, such as interfaces, inheritance, file handling, and user interfaces. This result suggests that fine-grained timestamp data is not actually necessary to identify programmers from their typing patterns. Only categorizing average keystroke latencies into three buckets – slow, mediocre, fast – might be enough for reliable identification.

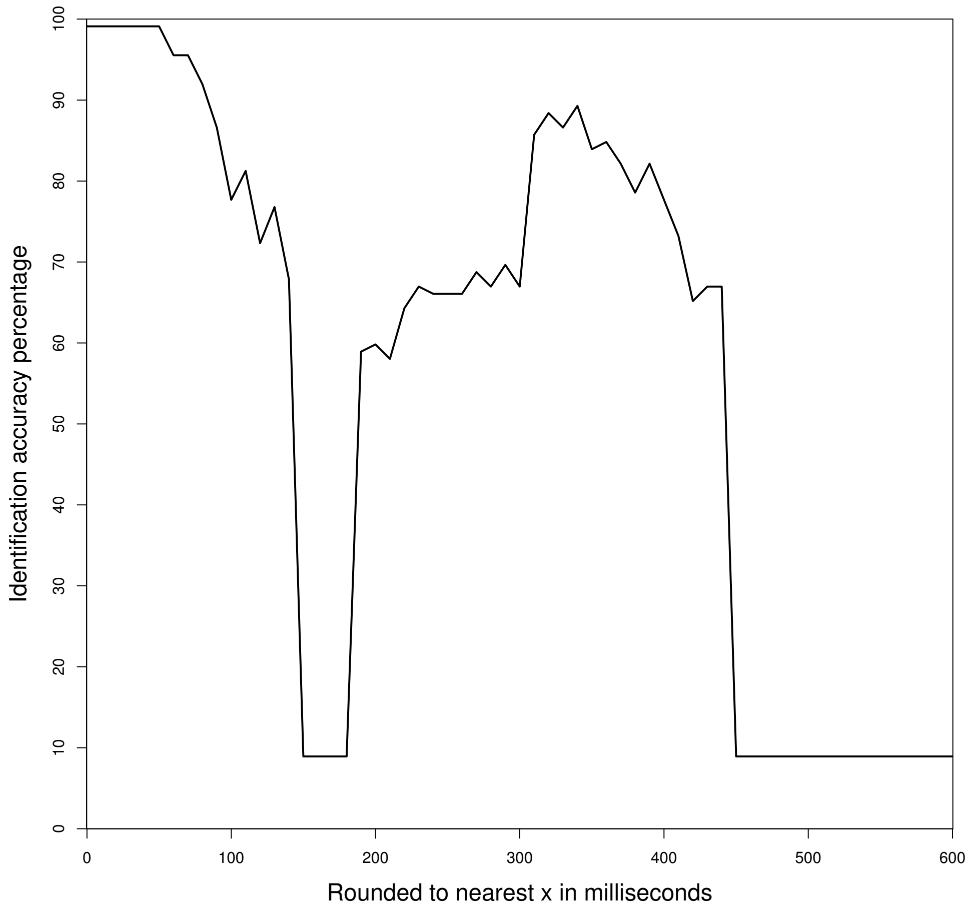

Figure 10: Identification accuracy compared against rounding precision. All values in the data were rounded to a nearest millisecond value. The larger the millisecond value in the x-axis, the lesser the rounding precision. The y-axis expresses identification accuracy.

Moreover, the observed effect is a cautionary result for researchers seeking to anonymize their data. Using a similar method and observing e.g. that the identification accuracies are low enough for sharing the data at the 200 millisecond point, and adding an additional 100 milliseconds "just to be sure", plenty of information that could be used to identify the individuals in the data would be shared accidentally.

6.3 Limitations

There are some limitations in our study, which we will address here. Firstly, we have only shown that the approaches covered in this work can prevent keystroke latency -based identification when our own previous identification method is used. It is possible that there are other methods for which neither rounding or bucketing the keystroke latencies prevents identification. In addition, we only show that keystroke latencies can not be used for identification when data has been de-identified with our methods. There can be, and most likely are, other identifiers in the data. Most obviously, explicit identifiers such as student numbers must also be removed from any data that is to be shared.

Even if identifying metadata such as the student numbers are removed, it is possible that subjects can be identfied based on the text content – both through explicit and quasi-identifiers in it. A good example is the source code snapshot data used in the experiments – the very first exercise the students complete on the course is to print out their name. While many students only print their first name, or a made-up name, some print their full name. This is an example of an explicit identifier in the text content. Furthermore, there can be quasi-identifiers in the text content. For example, the class and variable names a student uses could exhibit some patterns which could be used for identification as long as there is enough data. In addition, comments in the code could be analyzed for linguistic features that could be used for identification [14].

We chose programming experience as the valuable information we infer from the anonymized data. While we show that programming experience can still be inferred when the data is anonymized, it is possible that the value of the data has degraded in other aspects. It could be possible that the effect discussed in the beginning of Section 6.2, i.e. two possible values for features being enough to infer programming experience, is somehow specific to either our context or programming experience inference, and does not generalize to inferring other information. For example, Daries et al. [17] analyzed demographic data from MOOCs and noticed large differences in the results of their analysis when comparing non-anonymized data with anonymized data. They suggest that instead of anonymizing data, policies that protect the privacy of the subjects in the data should be created. They suggest for example that researchers who are given access to data sets are only allowed to conduct research where the primary objective is not to identify the individuals, and that the researchers are ethically and legally bound to not identify individuals even if it would be possible based on the data.

However, we consider that the experiments conducted in this work are a first step towards opening data sets for keystroke latency -based replication studies. We showed that there is a case – programming experience inference – where valuable information is retained in the data after anonymizing the keystroke latencies, which removed a quasi-identifier from the data.

7 Conclusions and Future Work

In this work, we studied two anonymization methods for de-identification of keystroke latency data. We had source code snapshot data from introductory programming courses held at the University of Helsinki in 2014 and 2015. The source code snapshots were processed into typing profiles. Previous research has shown that typing profiles built from source code snapshots can be used for identifying the typists [47]. Thus, we studied how identification accuracy is affected by modifying the keystroke latencies in the typing profiles. We used the identification model by Longi et al. [47] for the identification experiments. The objective was to modify the typing profiles so that identification accuracy would deteriorate. However, anonymization is only sensible if the anonymized data is valuable in some respect as otherwise it would not make sense to anonymize it in the first place. Hence, we further studied how inferring other information than the identity is affected by our anonymization procedures. As we have previously shown that typing profiles can be used to infer the programming experience of the programmers [42], we chose to reproduce those studies using different degrees of anonymization as a case study of whether valuable information (now programming experience) could be inferred from anonymized typing profiles.

The first approach to anonymization we studied was rounding all values in the data to a multiple of some millisecond value. The original values were average latencies for typing specific digraphs, i.e. character pairs. We found that rounding can be used for anonymizing keystroke latency data, but the amount of rounding needs to be sufficiently large in order to decrease identification accuracy. We discovered that whilst identification of students is no longer possible, programming experience can be inferred with almost similar accuracy compared to non-anonymized data. The second approach we studied was anonymizing data by distributing all values in the data into even-sized buckets. Both identification and programming experience classification accuracies decreased considerably faster than with the rounding method. The result therefore indicates that the bucket method is more efficient at anonymizing the data, although more domain relevant information is lost in the process – when the data was sufficiently anonymized, i.e. students could no longer be reliably identified, classification of students into novices and more experienced programmers was not as accurate as with the rounding method. The results for both anonymization techniques suggest that the anonymized typing profiles are no longer a quasi-identifier and thus do not pose a threat to the privacy of the individuals in the data. This shows promise of being able to share source code snapshot data openly in the future as long as other possible explicit and quasi-identifiers are likewise removed or anonymized adequately. Sharing data would facilitate replication studies, which in turn would increase the reliability and generalizability of results. However, careful consideration should be given to anonymization.

For example, we noticed that the decrease in identification accuracy is not linear when using the rounding method for anonymization. We propose that researchers looking into anonymizing their data should not settle for the first anonymization technique or degree that seems sufficient, but analyze their choices in great detail to guarantee that they are not merely observing a local minimum due to the characteristics of the chosen anonymization procedure.

Future work should examine how the methodologies outlined in this work perform with other identification models, for example other keystroke latency -based models that are not based on euclidean distances and the k-Nearest-Neighbor classifier. In addition, further research is needed to investigate whether other information than programming experience can be inferred from anonymized keystroke latency data. Finally, future work should investigate how removing possible hidden identifiers other than keystroke latencies – such as text content – affect both identification accuracy and inference of valuable information.

Acknowledgements

I thank Petri Ihantola for the inspirational discussion on bucketing keystroke latency data for anonymization purposes at Koli Calling '15 conference. In addition, I thank Arto Hellas and Arto Klami for guidance and feedback on this work, Krista Longi for collaboration on the previous studies [42, 47] without which this study would not have been possible to conduct, and my other colleagues in the Agile Education research group.

References

[1] Ahadi, Alireza, Lister, Raymond, Haapala, Heikki, and Vihavainen, Arto: *Exploring machine learning methods to automatically identify students in need of assistance.* In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 121–130, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-3630-7. `http://doi.acm.org/10.1145/2787622.2787717`.

[2] Ahadi, Alireza, Lister, Raymond, Haapala, Heikki, and Vihavainen, Arto: *Exploring machine learning methods to automatically identify students in need of assistance.* In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 121–130, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-3630-7. `http://doi.acm.org/10.1145/2787622.2787717`.

[3] Al-Zubidy, Ahmed, Carver, Jeffrey C, Heckman, Sarah, and Sherriff, Mark: *A (updated) review of empiricism at the sigcse technical symposium.* In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 120–125. ACM, 2016.

[4] Ben-Gal, Irad: *Bayesian networks.* Encyclopedia of statistics in quality and reliability, 2007.

[5] Bennedsen, Jens and Caspersen, Michael E: *Assessing process and product: A practical lab exam for an introductory programming course 1.* Innovation in Teaching and Learning in Information and Computer Sciences, 6(4):183–202, 2007.

[6] Bergadano, F., Gunetti, D., and Picardi, C.: *Identity verification through dynamic keystroke analysis.* Intell. Data Anal., 7(5):469–496, October 2003, ISSN 1088-467X. `http://dl.acm.org/citation.cfm?id=1293861.1293866`.

[7] Bergadano, Francesco, Gunetti, Daniele, and Picardi, Claudia: *User authentication through keystroke dynamics.* ACM Trans. Inf. Syst. Secur., 5(4):367–397, November 2002, ISSN 1094-9224. `http://doi.acm.org/10.1145/581271.581272`.

[8] Bergin, Susan and Reilly, Ronan: *Programming: factors that influence success.* ACM SIGCSE Bull., 37(1):411–415, 2005.

[9] Bixler, Robert and D'Mello, Sidney: *Detecting boredom and engagement during writing with keystroke analysis, task appraisals, and stable*

traits. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13, pages 225–234, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-1965-2. `http://doi.acm.org/10.1145/2449396.2449426`.

[10] Breiman, Leo: *Random forests.* Machine Learning, 45(1):5–32, 2001.

[11] Brown, Neil Christopher Charles, Kölling, Michael, McCall, Davin, and Utting, Ian: *Blackbox: a large scale repository of novice programmers' activity.* In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 223–228. ACM, 2014.

[12] Cepeda, Nicholas J, Pashler, Harold, Vul, Edward, Wixted, John T, and Rohrer, Doug: *Distributed practice in verbal recall tasks: A review and quantitative synthesis.* Psychological bulletin, 132(3):354, 2006.

[13] Cepeda, Nicholas J, Vul, Edward, Rohrer, Doug, Wixted, John T, and Pashler, Harold: *Spacing effects in learning a temporal ridgeline of optimal retention.* Psychological science, 19(11):1095–1102, 2008.

[14] Chaski, Carole E: *Empirical evaluations of language-based author identification techniques.* Forensic Linguistics, 8:1–65, 2001.

[15] Cho, Sungzoon, Han, Chigeun, Han, Dae Hee, and Kim, Hyung Il: *Web-based keystroke dynamics identity verification using neural network.* Journal of organizational computing and electronic commerce, 10(4):295–307, 2000.

[16] Coursera: *Coursera signature track.* `https://www.coursera.org/signature/`, 2016. Accessed: 2016-10-24.

[17] Daries, Jon P, Reich, Justin, Waldo, Jim, Young, Elise M, Whittinghill, Jonathan, Ho, Andrew Dean, Seaton, Daniel Thomas, and Chuang, Isaac: *Privacy, anonymity, and big data in the social sciences.* Communications of the ACM, 57(9):56–63, 2014.

[18] Dataverse: *The dataverse project.* `http://dataverse.org/`, 2016. Accessed: 2016-10-24.

[19] Dempster, Frank N: *The spacing effect: A case study in the failure to apply the results of psychological research.* American Psychologist, 43(8):627, 1988.

[20] Dowland, Paul S. and Furnell, Steven M.: *A long-term trial of keystroke profiling using digraph, trigraph and keyword latencies.* In Deswarte, Yves, Cuppens, Frédéric, Jajodia, Sushil, and Wang, Lingyu (editors): *Security and Protection in Information Processing Systems*, volume 147 of *IFIP - The International Federation for Information*

Processing, pages 275–289. Springer, 2004, ISBN 978-1-4757-8016-1. `http://dx.doi.org/10.1007/1-4020-8143-X_18`.

[21] Dwork, Cynthia: *Differential privacy: A survey of results.* In *Theory and applications of models of computation*, pages 1–19. Springer, 2008.

[22] Epp, Clayton, Lippold, Michael, and Mandryk, Regan L.: *Identifying emotional states using keystroke dynamics.* In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 715–724, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0228-9. `http://doi.acm.org/10.1145/1978942.1979046`.

[23] Fung, Benjamin C. M., Wang, Ke, Chen, Rui, and Yu, Philip S.: *Privacy-preserving data publishing: A survey of recent developments.* ACM Comput. Surv., 42(4):14:1–14:53, 2010, ISSN 0360-0300. `http://doi.acm.org.libproxy.helsinki.fi/10.1145/1749603.1749605`.

[24] Gaines, R Stockton, Lisowski, William, Press, S James, and Shapiro, Norman: *Authentication by keystroke timing: Some preliminary results.* Technical report, 1980.

[25] Gunetti, Daniele and Picardi, Claudia: *Keystroke analysis of free text.* ACM Trans. Inf. Syst. Secur., 8(3):312–347, August 2005, ISSN 1094-9224. `http://doi.acm.org/10.1145/1085126.1085129`.

[26] Hagan, Dianne and Markham, Selby: *Does it help to have some programming experience before beginning a computing degree program?* ACM SIGCSE Bull., 32(3):25–28, 2000.

[27] Haider, S., Abbas, A., and Zaidi, A.K.: *A multi-technique approach for user identification through keystroke dynamics.* In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 2, pages 1336–1341 vol.2, 2000.

[28] Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer, Bernhard, Reutemann, Peter, and Witten, Ian H: *The weka data mining software: an update.* ACM SIGKDD explorations newsletter, 11(1):10–18, 2009.

[29] He, Yeye and Naughton, Jeffrey F: *Anonymization of set-valued data via top-down, local generalization.* Proceedings of the VLDB Endowment, 2(1):934–945, 2009.

[30] Hosseini, Roya, Vihavainen, Arto, and Brusilovsky, Peter: *Exploring problem solving paths in a java programming course.* 2014.

[31] Hovemeyer, David, Hellas, Arto, Petersen, Andrew, and Spacco, Jaime: *Control-flow-only abstract syntax trees for analyzing students' programming progress.* In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 63–72, New York, NY, USA, 2016. ACM, ISBN 978-1-4503-4449-4. `http://doi.acm.org/10.1145/2960310.2960326`.

[32] Idrus, Syed Zulkarnain Syed, Cherrier, Estelle, Rosenberger, Christophe, and Bours, Patrick: *Soft biometrics for keystroke dynamics: Profiling individuals while typing passwords.* Computers & Security, 45:147–155, 2014.

[33] Ihantola, Petri, Vihavainen, Arto, Ahadi, Alireza, Butler, Matthew, Börstler, Jürgen, Edwards, Stephen H., Isohanni, Essi, Korhonen, Ari, Petersen, Andrew, Rivers, Kelly, Rubio, Miguel Ángel, Sheard, Judy, Skupas, Bronius, Spacco, Jaime, Szabo, Claudia, and Toll, Daniel: *Educational data mining and learning analytics in programming: Literature review and case studies.* In *Proceedings of the 2015 ITiCSE on Working Group Reports*, ITICSE-WGR '15, pages 41–63, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-4146-2. `http://doi.acm.org/10.1145/2858796.2858798`.

[34] Jadud, Matthew C: *Methods and tools for exploring novice compilation behaviour.* In *Proceedings of the second international workshop on Computing education research*, pages 73–84. ACM, 2006.

[35] Joyce, Rick and Gupta, Gopal: *Identity authentication based on keystroke latencies.* Communications of the ACM, 33(2):168–176, 1990.

[36] Karnan, M., Akila, M., and Krishnaraj, N.: *Biometric personal authentication using keystroke dynamics: A review.* Applied Soft Computing, 11(2):1565 – 1573, 2011, ISSN 1568-4946. `http://www.sciencedirect.com/science/article/pii/S156849461000205X`, The Impact of Soft Computing for the Progress of Artificial Intelligence.

[37] Kellaris, Georgios and Papadopoulos, Stavros: *Practical differential privacy via grouping and smoothing.* In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 301–312. VLDB Endowment, 2013. `http://dl.acm.org/citation.cfm?id=2488335.2488337`.

[38] Killourhy, Kevin S. and Maxion, Roy A.: *Free vs. transcribed text for keystroke-dynamics evaluations.* In *Proceedings of the 2012 Workshop on Learning from Authoritative Security Experiment Results*, LASER '12, pages 1–8, New York, NY, USA, 2012. ACM,

ISBN 978-1-4503-1195-3. `http://doi.acm.org/10.1145/2379616.2379617`.

[39] Koedinger, Kenneth R, Baker, Ryan SJd, Cunningham, Kyle, Skogsholm, Alida, Leber, Brett, and Stamper, John: *A data repository for the edm community: The pslc datashop*. Handbook of educational data mining, 43, 2010.

[40] Kurhila, Jaakko and Vihavainen, Arto: *Management, structures and tools to scale up personal advising in large programming courses*. In *Proceedings of the 2011 conference on Information technology education*, pages 3–8. ACM, 2011.

[41] Leinonen, Juho, Longi, Krista, Klami, Arto, Ahadi, Alireza, and Vihavainen, Arto: *Typing patterns and authentication in practical programming exams*. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 160–165. ACM, 2016.

[42] Leinonen, Juho, Longi, Krista, Klami, Arto, and Vihavainen, Arto: *Automatic inference of programming performance and experience from typing patterns*. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 132–137, New York, NY, USA, 2016. ACM, ISBN 978-1-4503-3685-7. `http://doi.acm.org/10.1145/2839509.2844612`.

[43] Leppänen, Leo, Leinonen, Juho, and Hellas, Arto: *Pauses and spacing in learning to program*. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, Koli Calling '16, pages 41–50, New York, NY, USA, 2016. ACM, ISBN 978-1-4503-4770-9. `http://doi.acm.org/10.1145/2999541.2999549`.

[44] Leppänen, Leo, Leinonen, Juho, and Vihavainen, Arto: *Short pauses while studying considered harmful*. In *EDULEARN 2016*, pages 1900–1904. IATED, 2016.

[45] Liu, Dapeng, Xu, Shaochun, and Cui, Zengdi: *Programmer's performance with the keystroke as an indicator: A further study*. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 577–583. IEEE, 2012.

[46] Liu, Huafu, Liu, Dapeng, and Xu, Shaochun: *Evaluating programming performance with keystroke characteristics: An empirical experiment*. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on*, pages 329–335. IEEE, 2013.

[47] Longi, Krista, Leinonen, Juho, Nygren, Henrik, Salmi, Joni, Klami, Arto, and Vihavainen, Arto: *Identification of programmers from typing patterns.* In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 60–67, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-4020-5. `http://doi.acm.org/10.1145/2828959.2828960`.

[48] Monaco, John V, Stewart, John C, Cha, Sung Hyuk, and Tappert, Charles C: *Behavioral biometric verification of student identity in online course assessment and authentication of authors in literary works.* In *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*, pages 1–8. IEEE, 2013.

[49] Monaco, John V and Tappert, Charles C: *Obfuscating keystroke time intervals to avoid identification and impersonation.* In *The 9th IAPR International Conference on Biometrics (ICB)*. IEEE, 2016.

[50] Monaco, J.V., Bakelman, N., Cha, Sung Hyuk, and Tappert, C.C.: *Recent advances in the development of a long-text-input keystroke biometric authentication system for arbitrary text input.* In *Intelligence and Security Informatics Conference (EISIC), 2013 European*, pages 60–66, Aug 2013.

[51] Monaco, J.V., Stewart, J.C., Cha, Sung Hyuk, and Tappert, C.C.: *Behavioral biometric verification of student identity in online course assessment and authentication of authors in literary works.* In *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*, pages 1–8, Sept 2013.

[52] Monrose, Fabian and Rubin, Aviel: *Authentication via keystroke dynamics.* In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 48–56, New York, NY, USA, 1997. ACM, ISBN 0-89791-912-2. `http://doi.acm.org/10.1145/266420.266434`.

[53] Monrose, Fabian and Rubin, Aviel: *Authentication via keystroke dynamics.* In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 48–56, New York, NY, USA, 1997. ACM, ISBN 0-89791-912-2. `http://doi.acm.org/10.1145/266420.266434`.

[54] Monsell, Stephen: *Task switching.* Trends in cognitive sciences, 7(3):134–140, 2003.

[55] Narayanan, Arvind and Shmatikov, Vitaly: *Robust de-anonymization of large sparse datasets.* In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 111–125, Washington, DC, USA,

2008. IEEE Computer Society, ISBN 978-0-7695-3168-7. http://dx.doi.org/10.1109/SP.2008.33.

[56] Pang, Ruoming, Allman, Mark, Paxson, Vern, and Lee, Jason: *The devil and packet trace anonymization.* SIGCOMM Comput. Commun. Rev., 36(1):29–38, January 2006, ISSN 0146-4833. http://doi.acm.org/10.1145/1111322.1111330.

[57] Pärtel, Martin, Luukkainen, Matti, Vihavainen, Arto, and Vikberg, Thomas: *Test my code.* International Journal of Technology Enhanced Learning 2, 5(3-4):271–283, 2013.

[58] Peacock, Alen, Ke, Xian, and Wilkerson, Matthew: *Typing patterns: A key to user identification.* IEEE Security & Privacy, 2(5):40–47, 2004, ISSN 1540-7993.

[59] Petersen, Andrew, Spacco, Jaime, and Vihavainen, Arto: *An exploration of error quotient in multiple contexts.* In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 77–86, New York, NY, USA, 2015. ACM, ISBN 978-1-4503-4020-5. http://doi.acm.org/10.1145/2828959.2828966.

[60] Powers, David Martin: *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.* 2011.

[61] Rivers, Kelly and Koedinger, Kenneth R: *Automating hint generation with solution space path construction.* In *International Conference on Intelligent Tutoring Systems*, pages 329–339. Springer, 2014.

[62] Rivers, Kelly and Koedinger, Kenneth R: *Data-driven hint generation in vast solution spaces: a self-improving python programming tutor.* International Journal of Artificial Intelligence in Education, pages 1–28, 2015.

[63] Rodrigo, Maria Mercedes T, Tabanao, Emily, Lahoz, Ma Beatriz E, and Jadud, Matthew C: *Analyzing online protocols to characterize novice Java programmers.* Philippine Journal of Science, 138(2):177–190, 2009.

[64] Samarati, Pierangela and Sweeney, Latanya: *Generalizing data to provide anonymity when disclosing information (abstract).* In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 188–, New York, NY, USA, 1998. ACM, ISBN 0-89791-996-3. http://doi.acm.org.libproxy.helsinki.fi/10.1145/275487.275508.

[65] Stewart, J.C., Monaco, J.V., Cha, Sung Hyuk, and Tappert, C.C.: *An investigation of keystroke and stylometry traits for authenticating online test takers.* In *Biometrics (IJCB), 2011 Int. Joint Conference on*, pages 1–7, Oct 2011.

[66] Sun, Yan and Upadhyaya, Shambhu: *Secure and privacy preserving data processing support for active authentication.* Information Systems Frontiers, 17(5):1007–1015, 2015, ISSN 1572-9419. `http://dx.doi.org/10.1007/s10796-015-9587-9`.

[67] Sweeney, Latanya: *k-anonymity: A model for protecting privacy.* International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10(05):557–570, 2002.

[68] Thomas, Richard C, Karahasanovic, Amela, and Kennedy, Gregor E: *An investigation into keystroke latency metrics as an indicator of programming performance.* In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, pages 127–134. Australian Computer Society, Inc., 2005.

[69] Vihavainen, Arto: *Predicting students' performance in an introductory programming course using data from students' own programming process.* In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on.* IEEE, 2013.

[70] Vihavainen, Arto, Paksula, Matti, and Luukkainen, Matti: *Extreme apprenticeship method in teaching programming for beginners.* In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 93–98. ACM, 2011.

[71] Vihavainen, Arto, Vikberg, Thomas, Luukkainen, Matti, and Pärtel, Martin: *Scaffolding students' learning using test my code.* In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 117–122. ACM, 2013.

[72] Villani, M., Tappert, C., Ngo, Giang, Simone, J., Fort, H.St., and Cha, Sung Hyuk: *Keystroke biometric recognition studies on long-text input under ideal and application-oriented conditions.* In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06. Conference on*, pages 39–39, June 2006.

[73] Vizer, Lisa M., Zhou, Lina, and Sears, Andrew: *Automated stress detection using keystroke and linguistic features: An exploratory study.* Int. J. Hum.-Comput. Stud., 67(10):870–886, October 2009, ISSN 1071-5819. `http://dx.doi.org/10.1016/j.ijhcs.2009.07.005`.

[74] Watson, Christopher, Li, Frederick WB, and Godwin, Jamie L: *Predicting performance in an introductory programming course by logging and analyzing student programming behavior.* In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 319–323. IEEE, 2013.

[75] Watson, Christopher, Li, Frederick WB, and Godwin, Jamie L: *No tests required: comparing traditional and dynamic predictors of programming success.* In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 469–474. ACM, 2014.

[76] Yu, Enzhe and Cho, Sungzoon: *Ga-svm wrapper approach for feature subset selection in keystroke dynamics identity verification.* In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 2253–2257, July 2003.