RESEARCH-ARTICLE

Raising Awareness for Inertial Sensors-based Keylogging on Smartphones

FEDERICO MONTORI, University of Bologna, Bologna, BO, Italy

LUCA SCIULLO, University of Bologna, Bologna, BO, Italy

LUCA BEDOGNI, University of Modena and Reggio Emilia, Modena, MO, Italy

Open Access Support provided by:

University of Bologna

University of Modena and Reggio Emilia

Published: 04 September 2024

Citation in BibTeX format

GoodIT '24: International Conference on Information Technology for Social Good
September 4 - 6, 2024
Bremen, Germany

Conference Sponsors:
SIGCAS

Raising Awareness for Inertial Sensors-based Keylogging on Smartphones

Federico Montori
federico.montori2@unibo.it
University of Bologna
Bologna, Italy

Luca Sciullo
luca.sciullo@unibo.it
University of Bologna
Bologna, Italy

Luca Bedogni
luca.bedogni@unimore.it
University of Modena and Reggio Emilia
Modena, Italy

ABSTRACT

Nowadays, inertial sensors are embedded in almost every smartphone and are a key enabler for a wide variety of applications that build on motion. However, in the context of major mobile operating systems, these sensors do not require any permission to be used. This may cause privacy and security breaches, as motion sensors can infer a multitude of derived conditions. Our paper aims to bring attention to keylogging through inertial sensors, in which they are used to understand what the user is typing building on how the device moves or tilts. We propose a pipeline for detecting whole words by applying a combination of supervised and unsupervised methods, to identify portions of the keyboards that display similar sensor values. We then combine this method further with word frequencies in a corpus to improve the detection accuracy. We performed a data gathering campaign by distributing a mobile app to multiple users and built up a real world dataset which we used to evaluate our proposal.

CCS CONCEPTS

• **Human-centered computing** → **Smartphones**; • **Security and privacy** → **Privacy protections**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Keylogging, Inertial sensors

ACM Reference Format:
Federico Montori, Luca Sciullo, and Luca Bedogni. 2024. Raising Awareness for Inertial Sensors-based Keylogging on Smartphones. In *International Conference on Information Technology for Social Good (GoodIT '24), September 04–06, 2024, Bremen, Germany.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3677525.3678634

1 INTRODUCTION

In recent years, the proliferation of smart devices equipped with inertial sensors, such as smartphones, tablets, and smartwatches, has led to a significant interest in leveraging these sensors for various applications beyond their primary scope. Among these applications, the detection of keyboard events through inertial sensors has emerged as a novel and intriguing area of research, which presents possibly severe user privacy implications [8]. Inertial sensors, including accelerometers and gyroscopes, offer precise data about vibrations. These sensors are capable of recording small movements of the user, who tilts the smartphone according to the position of the pressed character while typing.

One of the key motivations for exploring inertial sensor-based keyboard event detection is to enhance the privacy of users: inertial sensors can be used to identify abnormal typing patterns so that unauthorized access can also be detected and prevented. Moreover, they can also be utilized to monitor and analyze typing behaviors without requiring access to the actual content being typed, hence offering a privacy-preserving method for behavioral biometrics. This is particularly valuable in protecting sensitive information from being directly captured or logged.

However, this technology may also pose privacy risks, since the ability to infer keystrokes from sensor data raises concerns about unauthorized monitoring and the potential for keylogging user events and sentences. Specifically, these sensors currently do not require any permission to be used by common mobile operating systems. Hence, there is a need to raise awareness of the need to protect the privacy of users by highlighting potential breaches.

In this work, we present a study aimed at assessing the performance of key event detection on smartphone applications by using inertial sensors. Specifically, we first propose a preliminary study that shows the limitations of plain Machine Learning (ML) algorithms on this kind of sensor data; then, we present a pipeline for the detection of complete words typed by the user by using unsupervised methods and dictionary attacks on top of supervised vanilla ML models. We then validate the pipeline on a real world dataset that we gathered by developing an Android application that implements a small game and collects sensor data when the users type complete words.

The rest of this paper is structured as follows: Section 2 presents related works from literature; Section 3 describes our workflow and main contribution; Section 4 highlights the dataset we have collected; Section 5 presents results of testing the performance of ML classification algorithms on single letters, while Section 6 builds upon such classification to detect words; Section 7 concludes this work and discusses future lines of research.

2 RELATED WORKS

Detecting keyboard events on mobile devices requires the use of multiple sensor data, and it has been the target of several studies in the literature. The approach is similar to that used in Human Activity Recognition (HAR), as it requires gathering the data, and

GoodIT '24, September 04–06, 2024, Bremen, Germany

ACM ISBN 979-8-4007-1094-0/24/09
https://doi.org/10.1145/3677525.3678634

recognizing patterns on it that are representative of different human actions [7]. Many other works leverage these techniques to perform similar tasks [1], by combining sensor data with ML algorithms.

In general, sensor data on smartphones has also been considered a potential privacy issue, since it may also lead back to the user or device which created it. This is the focus of [12], which shows how to identify passwords on mobile devices leveraging data from accelerometers, or [10] which focuses instead on taps and swipe actions, or [4], which provides a high accuracy in identifying words thanks to a dictionary attack following the keystroke detection.

Using more than one device or more than one sensor on a single device has also been proven effective, since it is then possible to gather and fuse more information together, to capture more complex and hard-to-detect events [9] [6]. Other sensor data can be used, such as audio signals, which have been shown to be peculiar to some individuals and devices. The study presented in [2] is centered on mechanical keyboards, though other studies on mobile devices leverage a similar approach. It is also interesting the study presented in [5], which apart from inertial sensor data also leverages biometric signals from the user, such as the heart rate, to better model and detect the keystroke events.

What raises even more concerns is that most of this sensor data can be obtained on Android and iOS devices without the explicit permission of the user, passively by any application. Hence, a malicious application may exploit this data to obtain text, passwords, and in general private information without the user knowing it. There have also been studies centered on mitigation techniques for this issue. For instance [6] proposed to use noise in the sensor data, still providing an adequate precision of the data for the applications requiring it. Similarly, [11] introduces curated vibrations into the signal, so that they alter the inertial sensor data masking the real tap events. However, this may become disturbing for the users. We finally mention [3], which presents a similar technique to mask the original signal but does so in a context-aware manner. This means that depending on the context of use, it can introduce more or less noise according to the required sensitivity and to the risk of eavesdropping on private information about the user.

Compared to the body of literature, in this work we provide a comprehensive analysis of the features accountable for this kind of recognition, and test the limits and performance of the system with real users with a custom made Android application.

3 MAIN WORKFLOW

In this section we give a high-level view of our system in all its phases. The goal of our proposal is to perform a keylogging attack on a smartphone user, that is, obtain information on what the user is currently typing on a smartphone without her permission. Our system assumes to be unable to operate any software intrusion on the app the user is currently using, therefore a canonical keylogging attack is, in our case, unfeasible. Rather, we rely on one particular side channel that is available on smartphones: the inertial sensors.

Notably, neither on Android, nor on iOS phones there is a need for explicit permission in order to access inertial sensor data. As a result, any app installed on a smartphone can make deliberate use of accelerometers, gyroscopes, magnetometers, and all their software derivates (e.g. rotation sensor, gravity sensor, compass, etc...). Besides creating a potential leak of information that can lead to a loss of privacy, this also creates potential channels for attacks. Our aim here is to demonstrate that a keylogging attack is at least plausible by simply gaining access to inertial sensor data. Our approach builds on the combination of well-known supervised and unsupervised ML techniques; the challenges and the novelties in the process are in understanding how to combine these algorithms and how to interpret and use their results. Our approach goes through a number of phases that are depicted in detail in Figure 1. The rest of this section is dedicated to explaining the phases and their rationale, without delving into implementation details, as well as pointing out the current assumptions of this work.

3.1 Data Gathering

As mentioned earlier, our approach builds partially on supervised ML methods, which are necessarily dependent on training data. For this reason, we collected a large dataset that associates the action of a user typing a character on the smartphone keyboard with sensor readings occurring in the meantime. To this aim, we developed an Android application, called **SensRec**, which leverages a gamification approach to induce testers into typing a set of predetermined sentences. This approach sets up a very simple game: the user is displayed with a sentence and has to type it without mistyping (a mistake will simply make the user go back to the last correctly typed character). To make the game engaging, a timer is displayed, so that the user is encouraged to type the word as fast as possible. The application, at the same time, records the values of inertial sensors (such as accelerometer and gyroscope) and saves them into a CSV file. We will delve into technical depth and a more detailed description of the dataset in Section 4.

3.2 Preprocessing

At the end of the previous step, we end up with raw data in the form of multiple CSV files, where every line is a sensor reading associated with the key pressed by the user during the sensor reading (or a special wildcard in case nothing is pressed). First of all, in our approach, we take a data cleaning and feature calculation step. This involves deleting all sensor readings that do not occur while the user presses a key, as well as aggregating sensor readings in order to obtain a dataset where a single key pressed is associated with a number of features. This is done by, for each event in which a key is pressed, aggregating all its sensor readings and computing a set of statistical features (e.g. mean, variance, etc.).

Once this is done, the dataset is potentially ready to enter the ML pipeline, however, preliminary results showed not surprisingly quite poor performance. This is expected: detecting the key pressed on top of sensor values is inherently a hard classification task, due to the high number of classes and the quite similar sensor values. We expect in fact that two keys that are quite close in a keyboard to be extremely similar in terms of feature values, no matter the amount of features. Building on the intuition confirmed by our past work [4], we instead chose, at this stage, to focus on *areas of the keyboard* instead of the single characters. Indeed, if we can detect with reasonable accuracy the area of the keyboard that contains the pressed key, we can combine this with a dictionary attack and extract with a high probability the full word typed. In [4] we did

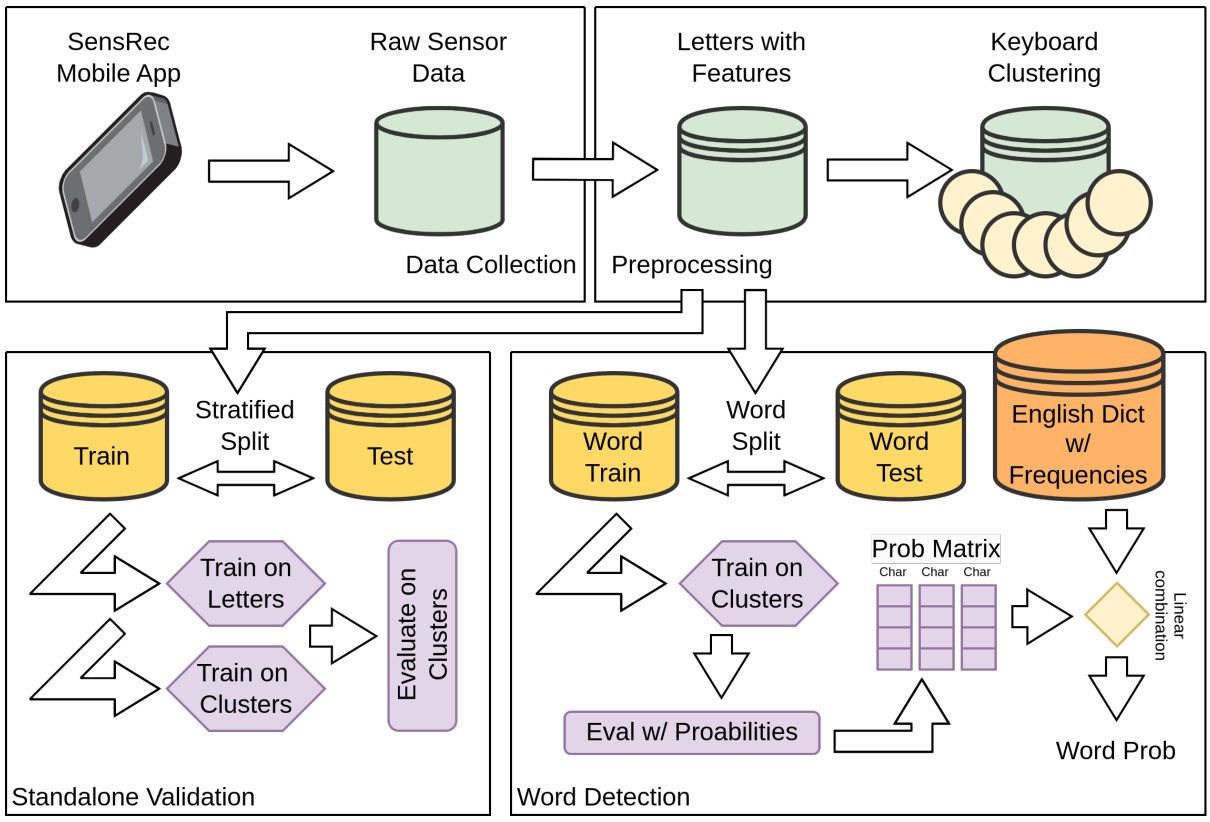


Figure 1: Main workflow of our solution

it by subdividing the keyboard into static square areas, however, despite the positive results, such a division is empirical and does not guarantee that keys within the same area would display a similar behavior. To overcome this limitation, in this work, we ran an unsupervised ML algorithm to cluster the keys in the keyboards according to their sensor values in k fixed sets. The value of k, by definition, is constrained in the interval $[1, K]$, where K is the number of keys in the keyboard used for the tests. In our case, we used the well-known K-means clustering to achieve this result. A more detailed description is given in Section 4.

3.3 Standalone Validation

The following step is to assess the validity of our intuition. The clustering described in the previous step is being executed for every possible value of $k \in \{1, \ldots, K\}$, in order to enable us to find the best possible value of k with respect to our goal. Intuitively, the higher the value of k, the smaller and more numerous the clusters will be, therefore yielding a lower accuracy. On the other hand, for small values of k, clusters are going to be big, which means that, even if the accuracy is high, we are still dealing with a high number of letters to choose from. Finding the optimal number of clusters, i.e., the optimal k, is crucial for the subsequent phases of the process. In this phase, we perform a standalone validation, i.e. on single keys, without taking into account full words. The dataset is split into training set and test set, then we train vanilla ML models on the training set. We here perform two kinds of training:

- **Train on letters**: here, models are trained in the usual way: every letter is considered as a different class.
- **Train on clusters**: here, models are trained by considering as a class the cluster label. Clearly, this process is performed for every value of k yielding exactly K models.

Next, we evaluate the models over the test set. The evaluation is done cluster-wise, thus we perform the testing for every single value of k using the cluster labels of the k-th configuration every time. If we are training on clusters, then the evaluation is straightforward, as the labels used in the training process are the same as in the testing. If, instead, we are training on letters, then the output of the prediction will be a letter, which we will have to map within the cluster labels in the k-th configuration used for the testing. This means that, for each value of k, we will need to keep in memory which keys belong to which cluster, as the clustering process is executed only once. This step outputs an accuracy value for each of the K values used for testing. By using graphical methods such as the *elbow method*, we can then empirically estimate the optimal number of clusters. More details are presented in Section 5.

3.4 Word Detection

Once the optimal number of clusters is determined, the process enters the last phase, where we try to detect whole words. The rationale behind this choice is that, as assessed by the previous phases, the correct detection of a single key is too difficult and uncertain. As a consequence, we build on the more confident detection of a single cluster as well as the presence of the word in a corpus of possible words that is significant for the use case. In our case, we used the English dictionary, as the sentences presented to the user in the SensRec app are all in plain English.

Consequently, we operate a different split of our dataset, which is based on words and allows no reshuffling. The training set is then used for training the ML model that best performs in the previous step, using the kind of training that appears to yield better results (among training on letters or clusters). The trained model is then used for evaluating every word in the test set. The word detection process takes place in the following way:

First, the model is tested on each letter composing the word to be tested, using, for each letter, the cluster label of the selected cluster number k^* (as an output from the previous phase). This model testing, for each letter composing the word, outputs an array of k^* probabilities, denoting the output probability of the model for the given cluster. This ultimately outputs a matrix $\mathcal{P}$ with shape $m \times k^*$, where m is the length of the word being evaluated. Next, we take the corpus of all possible words, and isolate all the words of length m; for each of them, we calculate their probability of matching the evaluated word through a linear combination. Specifically, let us define as $W = W_1, \ldots, W_m$ the evaluated word, with W_i identifying the cluster label of the i-th letter composing W, when using k^* clusters. Similarly, let us define as $W' = W'_1, \ldots, W'_m$ a word of the same length taken from the corpus. The probability of W' to match W is calculated as in Equation 1.

$$p(W, W') = \mathcal{P}_{1,W'_1} + \cdots + \mathcal{P}_{m,W'_m} \,. \tag{1}$$

It is easy to see that the formula above is the sum of the probability of the i-th letter of the word W' to be in the position i in W, such that $1 \leq i \leq m$.

This process outputs a probability for every word of length m to match the evaluated word W. We do not expect this process to identify W as the one with the highest probability, but that W is in a high percentile of all words of length m. This would be already a very good result, in fact it would, for instance, reduce the attempts for a brute-force attack by several orders of magnitude.

3.5 Known Limitations

The system presented in this section represents a complete pipeline, from data gathering to word detection. However, before delving into further details, it is important to outline that this methods builds on a number of assumptions that will need further research.

In this work, we assume to be able to recognize spaces, which allows us to come up with a perfect separation of the dataset into words in the word detection phase. This assumption allows us to focus more on word recognition, however, this should be taken into account when building the whole detected sentence. This aspect will be considered for future research.

We also assume that word detection is limited to the available corpus, which means that, for the moment, we do not allow the recognition of words that are outside our bag of knowledge. The inclusion of other words will be taken care of in subsequent studies.

Our approach for now accounts only for letters, without considering digits, punctuation, and other keystrokes that have additional effects (shift, enter, etc...). We are also not taking into account users who make mistakes while typing hence erase part of what is written and retype it. These are complex behaviors that deserve a separate study and, to the best of our knowledge, are an open problem.

4 BUILDING THE DATASET

In this section, we describe how we collected the dataset that we will then use in our experiments.

We developed an Android app, called SensRec, to collect data from a series of users. Specifically, SensRec listens for data obtained from the accelerometer, magnetometer, gravity sensor, and orientation sensor. To collect this data during typing events, each user should type sentences that appear on the screen, through a custom-developed keyboard that resembles the system keyboard. To limit the noise in the data, the sentence must be typed in a limited amount of time, so that no excessive vibrations are collected. If the time needed exceeds a threshold computed according to the length of the sentence, the data is rejected, and the same happens if the user types a wrong letter during the sentence.

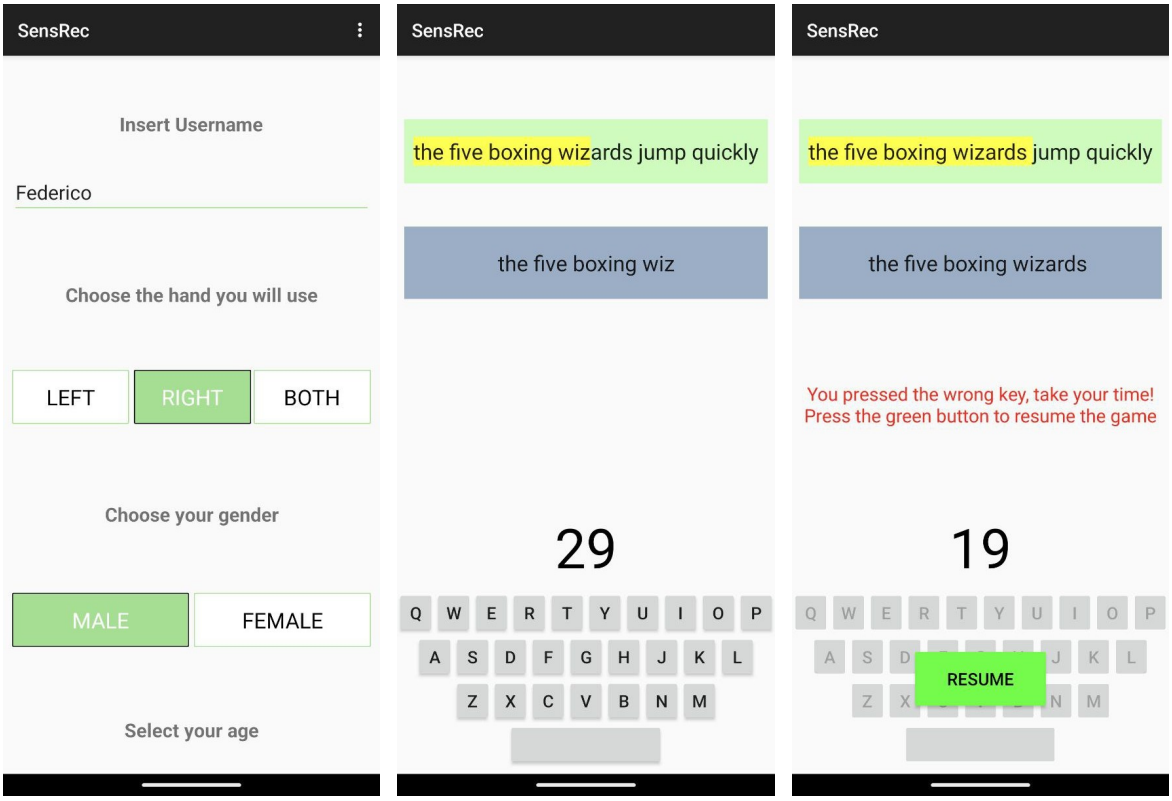
We save the data on a CSV file stored on the device, which is then uploaded to a Firebase storage to ease the retrieval of data from multiple devices. To complete the test, we required each recruited user to type at least 100 times each letter. We also report the statistics for each letter in a menu within the app, so that each user can keep track of their progress. The tests are performed using a single hand.

The application shows at first a screen where the user can type a pseudonym, so in case he/she has already performed any test, those will be considered from the same user. We also asked about the smartphone model, so that in the future we could see if there is a difference between different smartphone models in terms of data. We then ask the users about the hand they use to type the letters since the smartphone will be oriented differently depending on that. After filling in this information, the user is ready to start typing. A random sentence is sampled from a file containing 45 different sentences and displayed to the user, who can then start to complete the sentence. 15 of these sentences are pangrams, which means they contain all the letters of the alphabet – one of the most famous pangrams is "The quick brown fox jumps over the lazy dog" – so it is easier to end up with a balanced dataset.

We have collected data from 4 users, one with the left hand and three with the right hand, for a total of roughly 1,000 typed sentences. To build the final dataset, we have employed three different phases, described hereafter.

Adding the values of Magnitude, Roll and Pitch. In this first phase, we add three new columns to each record for each sensor we take into account, given that such sensors are triaxial. The magnitude represents the force obtained regardless of the orientation of the device, and it is computed depending on the inputs as

$$Mag(A_x, A_y, A_z) = \sqrt{A_x^2 + A_y^2 + A_z^2} \tag{2}$$



(a) Main menu (b) Typing in a sentence (c) Error in typing

Figure 2: Three screenshots of SensRec

$$Mag(A_x, A_y) = \sqrt{A_x^2 + A_y^2} \qquad Mag(A_x, A_z) = \sqrt{A_x^2 + A_z^2} \qquad Mag(A_y, A_z) = \sqrt{A_y^2 + A_z^2} \tag{3}$$

where A_i represents the i-th axis of sensor A.

We then add the Roll and Pitch, computed as

$$Roll(A_y, A_z) = \frac{A_y}{A_z} \qquad Pitch(A_x, A_y, A_z) = \frac{-A_x}{\sqrt{A_y^2 + A_z^2}} \tag{4}$$

For each sensor we then have 9 raw values, which are:

- The raw values along the axes x, y, and z
- The four magnitude values
- Roll and Pitch

Feature engineering. We then compute the features on each record. For each consecutive row in which we record the same key pressed, we aggregate them computing a series of statistical features, specifically the *Mean*, *Median*, *Standard Deviation*, *Variance*, *Max* and *Min*. These are computed for each value of each sensor, hence for each sensor we compute 54 different features.

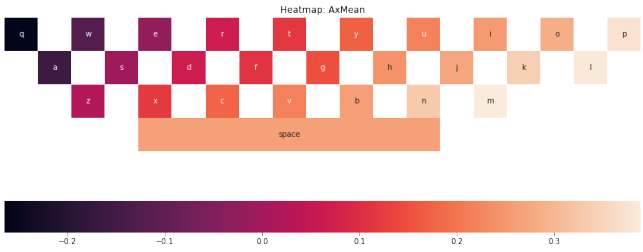
To highlight the differences between the features, we show two example heatmaps that represent the value of the specific feature over the keys. Precisely, Figure 3(a) represents the mean over the x axis of the accelerometer, while Figure 3(b) is the median value of the gyroscope y axis. Regardless of the specific values presented, it is possible to observe that they show very dissimilar trends, highlighting the benefits of having heterogeneous features that better describe each single key.

File merging. In the last step, we merge all the files from the different users in a single file. We have collected a dataset with more than 40,000 records considering both hands, which we publicly release[1].

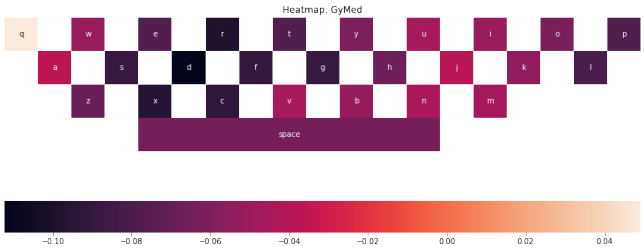
4.1 Clustering

Once the preliminary dataset is ready, we proceed to cluster the keys, hoping to isolate areas of the keyboard that have similar features, so that the detection of clusters would be easier than the mere detection of a single letter. In order to do that, as anticipated in Section 3.2, we implied the widely used K-means algorithm for this purpose. However, the sole application of K-means over the dataset does not guarantee that all samples of the same letter will end up in the same cluster, a condition that is necessary to come up with a unique and static "cluster map". For this reason, we employ the following method: from our dataset, we produced a second dataset by grouping all samples of the same letter into a single row in the new dataset. This aggregation produces a unique set of feature values for each letter obtained by averaging out all the occurrences of such values across the original dataset for such a

[1]https://doi.org/10.5281/zenodo.11490006



(a) Accelerometer over the x axis: Mean



(b) Gyroscope over the y axis: Median

Figure 3: Heatmaps of feature values for two different features. We can see how they yield different values depending on the key pressed.

letter. This not only ensures a unique 1:k mapping between letters and clusters, but also it ensures that no cluster is ever empty. Once this step is done, for each letter, the cluster label is associated with every sample in the dataset belonging to such cluster. Given that we executed K-means over all possible values of k, we now have, for each key in our dataset, K different labels. In particular, since we are using the English keyboard, only considering letters and dropping the spaces, we have that $K = 26$. By analyzing the corner cases, we observe that when $k = 1$ all samples will have the same label (as there is only one cluster), while when $k = K = 26$ then every letter will have a different label as if the clustering step was not performed. As a pictorial example, we show the output of this process when $k = 5$ in Figure 4.

5 SINGLE LETTER CLASSIFICATION

In this section, we will delve into details about the standalone classification step, introduced in Section 3.3 and its validation.

First of all, we separate the dataset into training set and test set observing the widely accepted 70% - 30% ratio, by stratifying the selection over letters. This ensures that the internal proportion between the occurrence of letters is kept in the two sets. Subsequently, we train seven different widely adopted ML models:

- Logistic Regression Classifier (LR)
- Gaussian Naïve Bayes Classifier (GNB)
- K-Nearest-Neighbors Classifier (KNN)
- C4.5 Decision Tree Classifier (DT)
- Support Vector Machines (SVM)
- Random Forest Classifier (RF)
- AdaBoost Classifier (AdaBoost)

Each model is trained K times, one for every cluster label associated with training examples, thus, we end up with 175 trained models – we did not train any model for $k = 1$. In Section 3.3 we anticipated how we both evaluate models trained on single letters (i.e. having the letter as a class label) and on clusters (i.e. having the cluster index as the class labels). In our case, training on single letters is equivalent to training on clusters with $k = K$. We evaluated the performance, in terms of accuracy, of all models trained both on letters and on clusters and tested on clusters for every possible value of k. Obviously, if training on clusters, then the value of k used for testing is the same used for training. The results of this experiment are depicted in Figure 5. They show, as expected, an almost monotone descending trend as the number of clusters increases in both cases. This is a natural consequence of an increasing number of classes. We also observe that, in both cases, the best performing algorithm overall is RF, with very few exceptions when k is low (i.e., $k = 2$, $k = 3$, and $k = 4$, the latter only if training on letters). In such cases, RF gets slightly outperformed by KNN and SVM, although by a negligible amount. Furthermore, it is clear that training on clusters yields globally slightly better results, at least for the best performing classifiers, making it the best candidate for subsequent steps. Finally, the elbow method privileges $k = 7$ in our case, as it shows an increase in performance with both training methods. That being said, the results promote using RF trained on clusters with $k = 7$ for the next phase.

6 WORD CLASSIFICATION

In this section, based on the premises in Section 3.4 and the results achieved in Section 5, we perform experiments with the goal of inferring whole words typed by the user. As anticipated, the rationale behind this choice comes from the results in Figure 5, which show that classic ML algorithms cannot achieve an accuracy higher than 0.4 on a single character, even excluding any character that is not a lowercase letter. The accuracy goes up to 0.6 if we choose $k = 7$, however, this result tells us that, for each letter, we have around 60% probability to have it belonging to a defined set, which counts two to five characters. This, taken alone, is still too uncertain.

Our intuition is that, if we restrict our detection to words that belong to a defined corpus, then we can combine classification probabilities of single characters and obtain a probability value for each word of the corpus. Because of the nature of the dataset, which is obtained by inducing users into typing words in plain English, we use as a corpus an English dictionary[2], which includes the 333,333 most common words derived from the Google Web Trillion Word Corpus alongside their frequency. We built our corpus by selecting the 80,000 most frequent words from there, as we empirically found that the likelihood of all other words is negligible.

First of all, we split our dataset differently from how it was done for the standalone experiment: in this case, we cannot shuffle letters, instead, we separate the dictionary into full words. The training set contains then 2,258 words counting a total of 11,740 letters, while the test set contains 958 words counting a total of 4,219 letters. Table 1 shows, for each possible value of word length m within the test set, the number of words both in the test set and in the English dictionary with that length.

[2]https://www.kaggle.com/datasets/rtatman/english-word-frequency

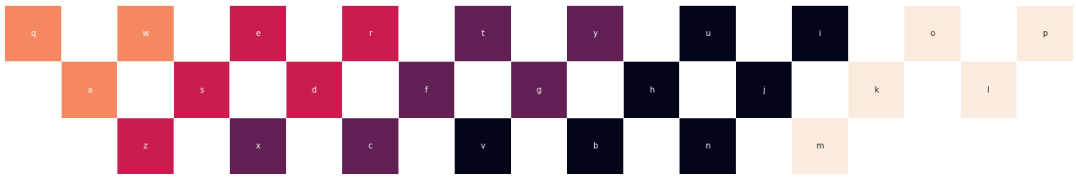
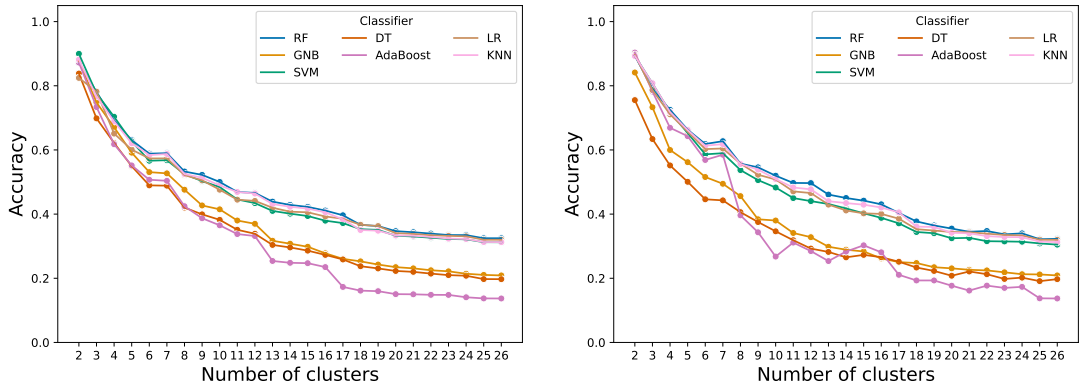


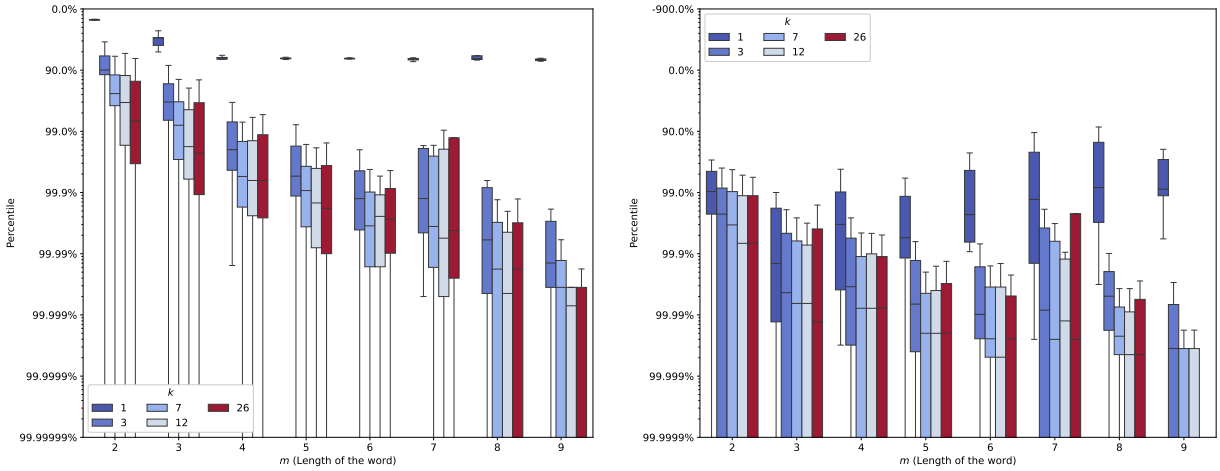
Figure 4: Example output of the clustering process by running K-means over the modified dataset with $k = 5$.



(a) Training on single keys

(b) Training on clusters

Figure 5: Standalone performance of supervised ML algorithms when considering different numbers of clusters.



(a) Plain probabilities

(b) Probabilities with word frequency

Figure 6: Position of the tested word within all words of the same length in the corpus expressed in percentile.

Word Length (m)	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
Words in the test set	48	121	202	232	106	66	68	40	44	15	8	0	3	953
Words in the corpus	26	676	6,590	7,503	9,333	11,314	11,620	10,380	8,296	6,023	3,660	2,151	1,248	78,793

Table 1: Number of words in the test set and in the corpus for different sizes

We then applied the procedure described in Section 3.4 on each word of the dataset, applying Equation 1 to each word of the same length in the corpus, to extract their probability to match the tested word. This results in us evaluating, for a tested word of length m, all possible words of length m in the English dictionary. We then sort all these words by their calculated probability in descending order and assess the position of our tested word in this list. The closer the word is to the top of the rank, the more accurate the prediction. Our evaluation metric is the percentile of this rank. In other words, if the set of words in the corpus with length m is C_m, and the tested word W matches with the j-th most probable word in C_m, then, the percentile of $W \in C_m$ is calculated as: $\frac{j}{C_m} \cdot 100$

The results of this procedure are shown in Figure 6(a), where we evaluate separately words of different length. We deliberately excluded $m = 1$ because the corpus is too small for such a length and the experiment would be exactly like a standalone evaluation. At the same time, we excluded values of $m \geq 10$ because of the few examples in the text set, which would make the assessment not significant. The y-axis is a reverse percentile (from 1 to 0) represented in logarithmic scale. We choose to evaluate the following values of k: $\{1, 3, 7, 12, 26\}$, in order to include the corner cases and a few representatives. From the results we observe that if we evaluate on single letters (i.e. $k = 26$) the results are slightly better and increasing in quality as m increases. For example, for $m = 2$ and $k = 26$ we can expect the probability of W to be in the top ~ 6 words, while for $m = 6$ and $k = 26$ the probability of W is usually in the first ~ 11. For smaller values of k the results are worse, which is expected, because all words of length m with their letters in the same clusters will have the same probability. If clusters are sufficiently big, then we are considering also words that are very unlikely to appear. This leads to our next intuition: if using clusters yields more accurate standalone results, then we might consider to take into account the likelihood of the word to appear based on a frequency attribute in the corpus. For example, our corpus assigns a frequency value to each word in the English dictionary. We conducted a preliminary test in which we simply modified Equation 1 as follows:

$$p(W, W') = f(W') \cdot (\mathcal{P}_{1,W'_1} + \cdots + \mathcal{P}_{m,W'_m}) \,, \tag{5}$$

where $f(W')$ is the frequency of W' as stated in the corpus.

The result of this second evaluation can be seen in Figure 6(b), which improves the performance of the previous evaluation by approximately an order of magnitude, as well as bringing the performance of other values of k in line with $k = 26$. With respect to the previous example, for $k = 26$ we can now expect the probability of W for both $m = 2$ and $m = 6$ to be the top word. This is a preliminary result, that puts in evidence the importance of considering the word frequency when a corpus is available, and that it is of particular value for medium values of k (the improvement over the previous experiment is much higher). Notably, we just multiplied the frequency by the probability without any tuning here. An important step forward in future works will be to adjust frequency and probability and study their balance over different datasets. Another takeaway message is that a simple dictionary attack would not be enough: in fact, if we look at the performance of $k = 1$ (where the probability is equal for every word and only the frequency is considered), the results settle around the 99th percentile, which for $m > 2$ means still choosing from few tens of words.

7 CONCLUSIONS

In this paper, we have presented our pipeline aimed at performing keylogging on smartphones by leveraging the inertial sensors. We first showed how standalone supervised ML algorithms perform poorly and then we proposed a method for detecting whole words by combining the former with clustering algorithms to identify keyboard areas that output similar sensor values. We tested our method on a real world dataset that we gathered and showed how combining our method with the probabilities derived from the word frequencies in a corpus would improve our results.

In Section 3.5 we acknowledged the limitations of our work. Our future line of research is directed to take out some assumptions, such as the detection of the spacebar, the detection of punctuation, and the correct handling of mistypes.

REFERENCES

[1] D. Anguita, Alessandro Ghio, L. Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. 2013. A Public Domain Dataset for Human Activity Recognition using Smartphones. In *The European Symposium on Artificial Neural Networks*. https://api.semanticscholar.org/CorpusID:6975432

[2] D. Asonov and R. Agrawal. 2004. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004.* 3–11. https://doi.org/10.1109/SECPRI.2004.1301311

[3] Adam J. Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M. Smith. 2012. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, Florida, USA) *(ACSAC '12)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/2420950.2420957

[4] Luca Bedogni, Andrea Alcaras, and Luciano Bononi. 2019. Permission-free keylogging through touch events eavesdropping on mobile devices. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 28–33.

[5] Andrea Bianchi, Ian Oakley, and Dong Soo Kwon. 2010. The secure haptic keypad: a tactile password system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1089–1092. https://doi.org/10.1145/1753326.1753488

[6] Liang Cai and Hao Chen. 2011. TouchLogger: inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX Conference on Hot Topics in Security* (San Francisco, CA) *(HotSec'11)*. USENIX Association, USA, 9.

[7] Claudia Carpineti, Vincenzo Lomonaco, Luca Bedogni, Marco Di Felice, and Luciano Bononi. 2018. Custom Dual Transportation Mode Detection By Smartphone Devices Exploiting Sensor Diversity. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 367–372. https://doi.org/10.1109/PERCOMW.2018.8480119

[8] Abdul Rehman Javed, Mirza Omer Beg, Muhammad Asim, Thar Baker, and Ali Hilal Al-Bayatti. 2023. Alphalogger: Detecting motion-based side-channel attack using smartphone keystrokes. *Journal of Ambient Intelligence and Humanized Computing* (2023), 1–14.

[9] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing* 5, 6 (2009), 657–675. https://doi.org/10.1016/j.pmcj.2009.07.007 PerCom 2009.

[10] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. 2011. (sp)iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '11)*. Association for Computing Machinery, New York, NY, USA, 551–562. https://doi.org/10.1145/2046707.2046771

[11] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. 2012. Tapprints: your finger taps have fingerprints. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (Low Wood Bay, Lake District, UK) *(MobiSys '12)*. Association for Computing Machinery, New York, NY, USA, 323–336. https://doi.org/10.1145/2307636.2307666

[12] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. 2012. ACCessory: password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications* (San Diego, California) *(HotMobile '12)*. Association for Computing Machinery, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/2162081.2162095