Towards Behaviometric Security Systems: Learning to Identify a Typist

Mordechai Nisenson[2], Ido Yariv[1], Ran El-Yaniv[1], and Ron Meir[2]

[1] Department of Computer Science
{yariv@vipe,rani@cs}.technion.ac.il
[2] Department of Electrical Engineering
Technion - Israel Institute of Technology
{sm0ti@t2,rmeir@ee}.technion.ac.il

Abstract. We consider the problem of identifying a user typing on a computer keyboard based on patterns in the time series consisting of keyboard events. We develop a learning algorithm, which can rather accurately learn to authenticate and protect users. Our solution is based on a simple extension of the well known Lempel-Ziv (78) universal compression algorithm. A novel application of our results is a second-layer *behaviometric security system*, which continually examines the current user without interfering with this user's work while attempting to identify unauthorized users pretending to be the user. We study the utility of our methods over a real dataset consisting of 5 users and 30 'attackers'.

1 Introduction

Many security systems rely on a single log-on entry, typically a password, for access. Such systems can be compromised if the password is discovered, or is easy to attack. Greater security is achieved by relying on a physical means of identification, most often an access card (which may also include a One-Time-Pad to generate secure passwords). But if the card is lost it too could become a security risk. In general, all of these systems are vulnerable to an attacker co-opting a user's session; either by physically taking the place of the user, or by some exploitable weakness in the system. Recently, biometric methods have begun to appear in widespread use (see e.g. [1]). These typically rely on fingerprints or retinal structure. While some biometric security methods are considered rather safe, by and large these systems are only used for single log-on, and require additional hardware.

A different class of identification methods can rely on patterns appearing in a user's behavior when interacting with a machine. Possible examples could be driving a car, or interacting with a computer through typing, mouse control, navigation patterns, and so on. Such *behaviometric identification* is different from biometric identification in two respects[1]. On the one hand, behaviometric measurements can be intentionally biased (or corrupted) to some extent by

[1] The field dealing with measurements, theories and analysis of patterns in all aspects of human behavior is called *behaviometrics*.

N. Lavrač et al. (Eds.): PKDD 2003, LNAI 2838, pp. 363–374, 2003.

users who can control their behavior. On the other hand, unlike biometric measurements, behaviometric readings can be done on a continuous basis without interrupting or interfering with users' activities. This possibility allows for creating a secondary security system, which is continually operated after log-on. Such a system need not only be applicable to computers but to a wider array of other devices as well; conceivably any device with a sufficiently complex input system.

The aim of this study is to examine the question of whether a behaviometric security method can be automatically learned by a machine. In particular, we focus on the problem of *typist identification*. While being a particular instance of behavior, we believe that typing can represent some essential and general issues in behaviometric identification. Like other types of interactions with machines it is suggested that every person types differently, not only having to deal with typing method (e.g. touch typing), but more importantly with a person's physical and mental attributes. The size of one's hands, length of one's fingers, fine motor skills, language skills, and knowledge of keyboard layout could all come into play to affect how one types. Thus, identifying a typist is an interesting and challenging problem worthy of behavior analysis.

Our solution to the learning of typist classifiers is based on a number of simple ideas, which combine well into an effective method. We represent sequences of typing events as discrete sequences over finite (and rather small) alphabets, and then use universal prediction machines (based on known universal compression algorithms) to generate probabilistic behaviometric models for users. Using these models we then solve instances of *single-class* classification problems. We describe the method and evaluate its performance over a real dataset collected from various typists. Our examination provides a proof of concept indicating that automatic learning of behaviometric identification of typists is a feasible task.

2 Problem Setup and Preliminaries

With a security application in view (as mentioned above), we model the typist identification problem as a *single-class* classification problem where we have a training set of typing samples from one user u and we would like to construct a classifier capable of distinguishing new typing sequences generated by u from sequences generated by other users. Usually, in this single-class setting the other samples (not from u) are referred to as *outliers*.

In general, a single-class classification formulation is required whenever it is possible to acquire training examples of the target class (e.g. typing sequences of the user u) but hard or impossible to collect examples of the outliers (e.g. sequences of intruders). Thus, while the desired classifier is still binary and should discriminate between the target and the outliers, only one side of the boundary is supported by the data. Therefore single-class classification problems are harder (and much less studied) than standard binary classification problems (see also Section 6). The performance of a single-class classifier is best measured using

standard statistical distinctions between error of the first type δ_1, giving the proportion of target samples which are classified as outliers, and error of the second type δ_2 measuring the proportion of outlier samples classified as target samples. A plausible requirement, in the context of security systems, is that the tradeoff between δ_1 and δ_2 is controlled by the user.

We now characterize more formally the typing sequences we consider. The output generated when a user operates a keyboard is a sequence of events which can be described as follows. Standard keyboards usually have 104 keys and the keyboard outputs *events* when keys are pressed and released. Let K be the set of keys on the keyboard (that is, $|K| = 104$). Each key can be in one of two states: *pressed* or *released*. Let $A = \{\text{press, release}\}$ be this set of states. A *keyboard event*, $e = (k, a)$, where $k \in K, a \in A$, occurs whenever a key is pressed or released. Let E be the set of all keyboard events. Clearly, $|E| = |K||A| = 2|K| = 208$. A sequence $e_1, e_2, \ldots, e_n$ of keyboard events is viewed as a *time series* $x_1, x_2, \ldots, x_n$ where $x_i = (e_i, t_i)$ and t_i is the time recorded for the event e_i. Any such finite time series of keyboard events is called a *sentence*.

3 Typist Identification via Universal Prediction

The proposed solution to typist identification is based on universal prediction algorithms for discrete sequences. In this section we first describe a transformation of input sentences into a suitable representation for the use of such prediction algorithms. We then describe the prediction algorithm, which is obtained by extending a standard Lempel-Ziv compression algorithm.

3.1 Representation via Quantized Time Differentials

As described above each input sample is a time sequence $(e_1, t_1), (e_2, t_2), ..., (e_n, t_n)$ of keyboard events. The exact times at which events take place are of little value. Of much greater interest is the time differential between two events. Not surprisingly (and as noted by others, e.g. [2]), these differentials contain much of the discriminative information between typists. Setting $\Delta_i = t_{i+1} - t_i$, we transform the sentence into a sequence of its differentials so that $(e_i, t_i) \rightarrow (e_i, \Delta_i, e_{i+1})$. The resulting sequence of triplets consisting of events and differentials faithfully represents the time transitions between events which are relevant to typist discrimination. However, we choose to use the following slightly different differential representation which can be uniquely determined from a triplet sequence.

$$(e_1, \Delta_1, e_2), \ldots, (e_{n-1}, \Delta_{n-1}, e_n) \leftrightarrow e_1, \Delta_1, e_2, \Delta_2, \ldots, e_{n-1}, \Delta_{n-1}, e_n.$$

This last representation (on the right-hand side) is simpler in the sense that it has a smaller "alphabet" size. However, while the number of events is finite the number of time differentials is not. First, unbounded differentials can be avoided by specifying that all values larger than a specific Δ_{max} represent the start of a new sentence (keystrokes that are minutes apart are unlikely to be

related in any fashion). Second, for a given a set of sentences D_u, emitted by the typist u, which are to be learned, an additional transformation on the time differentials is performed with the goal of limiting the number of time differentials and smoothing over them. Fewer symbols make the data easier to learn, by reducing statistical sparseness (and thus reducing variance). To accomplish this, vector quantization [3] is used to cluster the time differentials into Q clusters, with Q centroids $c_1, c_2, ...c_Q$. The time differentials then undergo the following transformation:

$$\Delta \Rightarrow c^* \quad \text{where} \quad c^* = \underset{c_i}{\operatorname{argmin}} |\Delta - c_i|.$$

This transformation is used on all sentences that are to be learned or ranked by the user's model. Thus, the final makeup of a sentence is $\{e_1, q_1, e_2, q_2, ..., q_{n-1}, e_n\}$, where $q_i \in \{c_1, \ldots, c_Q\}$ represents some time differential Δ. Therefore, the number of symbols in our alphabet is $|E| + Q$. Note that the number Q of centroids becomes a parameter of the algorithm.

3.2 Lempel-Ziv Universal Prediction

Having represented a keyboard event sequence as a sequence of discrete symbols over a finite alphabet, we can now use any universal prediction algorithm for discrete sequences to generate conditional likelihood estimates of unseen sequences. Specifically, given a set D_u of training sentences for user u, we use a universal prediction algorithm to train a model M_u which is then capable of estimating $\Pr(x|D_u)$, the conditional probability distribution of an unseen sentence x. Using such conditional estimates we then solve the single-class classification problem.

There are a number of universal prediction algorithms whose empirical performance for lossless text compressions is considered state-of-the-art. Notable examples are the context tree weighting method (CTW) [4], the Burrows-Wheeler Transform (BWT) (see e.g. [5]) and variants of Prediction by Partial Matching (PPM) [6]. For simplicity and for computational efficiency we compromise likelihood estimation accuracy and rely on the Lempel-Ziv algorithm (`lz78` [7]). In particular, we use the prediction component of the `lz78` algorithm as described in [8]. Besides being very simple and fast this algorithm enjoys performance guarantees of various types (see e.g. [9]). We also propose two improvements to the algorithm, which appear to increase its prediction accuracy.

The `lz78` Universal Prediction is a one-pass algorithm. It builds a weighted tree from sequences over a finite alphabet, and can assign probability estimates to new sentences given such a tree. The `lz78` phrase tree holds a "dictionary" of phrases parsed from the training sequence and is constructed by parsing input sequences as follows. At each stage the algorithm parses the smallest prefix which is not yet in the tree. For example, the string "ababbac" is parsed into: a, b, ab, ba, c. This set of phrases can be viewed as a phrase tree such that each parsed phrase is a path from the root to a leaf (see [8] for a detailed exposition).

As described in [8] the phrase tree can be extended to provide count statistics by adding a counter to each node. These count statistics can be used to calculate a probability estimate for traversing from a parent node to one of its children.

Given a set of sequences, $s_1, \ldots, s_k$, emitted by some source (say the user u), a parse tree with appropriate counter statistics can be constructed for all s_i (e.g. by concatenating the s_i into one long sequence). The resulting statistical model is denoted by M_u. M_u can be used to compute the conditional probability $\Pr(x|M_u) \approx \Pr(x|s_1, \ldots, s_k)$ of a new sequence x. This is done by traversing down from the root according to the letters of x, and multiplying the probability estimates of the traversals, until a leaf is reached. Then the traversal resumes from the root. In practice, the normalized (negative) log-likelihood is used,

$$V(x, M_u) = \frac{-\log_2 \Pr(x|M_u)}{|x|} \,. \tag{1}$$

This value is non-negative for all x and is 0 (for finite length strings) only when $Pr(x|M_u) = 1$, which is the ideal prediction for any sentence emitted by u.

3.3 Improvements to Standard LZ Prediction

A major advantage of the `lz78` parsing technique is its speed. This speed is possible by compromising a systematic consideration of all substrings. While for very large training sets this compromise will not affect the results significantly, for small training sets (and short test sequences) this results in sparser and noisier statistics. We propose two simple modifications to the algorithm which increase the number of phrases extracted and improve performance of the `lz78` estimation. The two modifications are termed *input shifting*, and *back-shift parsing*.

Input shifting is used during the learning process to extract more phrases from a sentence. Considering a sentence $x = x_1 x_2 \cdots x_n$, the sentence is parsed once as described above. Then it is parsed s more times, where in the ith additional parsing we parse the suffix $x_{i+1} x_{i+2} \cdots x_n$ in the usual way (but starting with the aggregated model constructed by previous parsings). The effect of input shifting is to increase the number of phrases thus making the phrase tree larger. As s grows so does the height of the tree as longer and longer phrases are parsed. Note that by taking $s = 0$ we leave the `lz78` algorithm intact.

Another deficiency of the `lz78` algorithm is the loss of context when parsing a sequence (and when calculating the likelihood of a sequence). Specifically, each time the algorithm returns to the root (see description in Section 3.2) after parsing a phrase in a sequence, the entire context consisting of previous symbols is lost. In order to remedy this, we propose a method which utilizes the last m letters parsed to provide a prior context for the next phrase (taking $m = 0$ leaves `lz78` intact). This method, which we term *Back-shift parsing (BSP)* seeks to achieve this by back-shifting m letters after parsing each phrase. This approach is problematic for $m > 1$, however, since more letters may be back-shifted than parsed (which occurs often in practice). This seriously impedes progress and compromises speed, which is one of the advantages of `lz78`.

We prevent this by requiring that the m letters come from the last phrase parsed. This slight change is implemented by utilizing a "marker" as described in Figure 1. The resulting Back-Shift Parsing with a marker prevents back-tracking beyond the marker thus guaranteeing rapid progress. The overall effect is to quickly build a tree with no path shorter than $m + 1$ in length, or to make the tree deeper while minimally affecting its width.

```
Initialization: marker = start of sentence

Repeat until no more phrases to be parsed:
     phrase = next phrase parsed
              (starting at marker)
     add phrase to dictionary
     if (length(phrase) > m)
        marker = marker + length(phrase) - m
```

Fig. 1. Pseudo-code for Back-Shift Parsing (BSP) with a marker.

BSP also affects the calculation of a probability estimate for an unseen sentence. Instead of returning to the root after traversing to a leaf, the last m letters traversed are first traced down from the root to some node v, and then the new traversal begins from v (if v does not exist, then the new traversal continues from the root instead).

The modified algorithm now has two parameters and is denoted by `lz78`(s, m) where s determines the number of input shifts and m determines the context length for back-shifting. The following example shows the parsed phrases generated by some `lz78`(s, m) algorithms for the sequence "`ababbac`". Note that the phrases appear in the order of their parsing.

Algorithm	*Phrases Parsed from "ababbac"*
lz78(0,0)	{a,b,ab,ba,c}
lz78(1,1)	{a,ab,b,ba,abb,bac,c,bab,bb}
lz78(2,2)	{a,ab,aba,b,ba,bab,abb,bb,bba,bac,ac,babb,bbac,abba}

3.4 Single-Class Classification and Model Selection

Let $D_u = \{S_1, \ldots, S_n\}$ be a training set of sentences emitted by u. Given a fixed choice of the parameters s and m we use the `lz78`(s, m) algorithm to build a model $M_u = M(D_u, s, m)$ for the user u (thus, for a particular user, a model corresponds to a choice of s and m). This model can provide likelihood estimates for unseen sentences. Given an unseen sequence x we should determine whether $\Pr(x|M)$ is sufficiently large to "accept" x (alternatively, that $V(x, M_u)$ is sufficiently small; see Eq. (1)). To this end, a cutoff point, or threshold t, is necessary. We determine a threshold using the following leave-one-out methodology. For each training sentence S_i in D_u we calculate the likelihood of S_i given a model trained on D_u excluding S_i. More formally, for each $S_i \in D_u$ let

$$V_i = V\left(S_i, M(D_u \setminus \{S_i\}, s, m)\right) \ . \tag{2}$$

Let μ_M and σ_M be the empirical average and standard deviation of the V_i, $i = 1, \ldots, n$. An ideal (but perhaps not achievable) threshold t places all (future) user's sentences below the cutoff and attackers' sentences above. Given the evidence we have (the training sentences for u) we attempt to guarantee results for the user by setting the threshold to $t(M) = \mu_M + k_\sigma \sigma_M$ where k_σ is

sufficiently large. Using Chebyshev's inequality, for any k_σ we can provide for u a *confidence level* as follows. For any random variable X whose mean and standard deviation are μ and σ, respectively, a one-tailed version of Chebyshev's inequality [10] states that for any $k \geq 0$, $\Pr\{X - \mu \geq k\sigma\} \leq \frac{1}{1+k^2}$. Considering future sentences emitted by u as observations of a random variable S and taking μ_M and σ_M as estimates of the true mean and standard deviation of the random variable $V(S, M(s, m, D_u))$, we have for any choice k_σ (and using $t(M) = \mu_M + k_\sigma \sigma_M$), $\Pr\{V(S, M(s, m, D_u)) > t(M)\} \leq \frac{1}{1+k_\sigma^2}$. Thus the confidence level is $1 - \delta = 1 - 1/(1 + k_\sigma^2) = k_\sigma^2/(1 + k_\sigma^2)$.

To summarize, our typist identification algorithm has four parameters: Q, the quantization level; m and s, the parameters of the improved `lz78` algorithm; and k_σ, which determines acceptance threshold. Our goal is to set values to these parameters based only on the training set D_u.

Within a minimax setting, we choose the best model which maximizes the likelihood of the "hardest" training sentence. Specifically, we take

$$M_u^* = \underset{M}{\operatorname{argmin}} \left\{ \max_{S_i \in D_u} V_i \right\} \tag{3}$$

This optimization determines values for the parameters Q, m and s. The parameter k_σ is set such that the maximum V_m value in (3) is just below the threshold $t(M_u^*)$ and will be accepted by the model. Specifically, $\max_{S \in D_u} V_M(S, D_u) = \mu_M + k_\sigma \sigma_M$ and solving for k_σ we get

$$k_\sigma = \frac{\max_{S \in D_u} V_{M_u^*}(S, D_u) - \mu_{M_u^*}}{\sigma_{M_u^*}}.$$

In addition to the above single-class setting we also consider a setting where a (small) set of "attacker" sentences is available for training. Clearly, if such a set of "outliers" is not very large, it is not likely to faithfully represent the general statistics of outliers. However, it is interesting to investigate whether this additional piece of information can be exploited to improve performance. Although this problem is typically not a standard two-class problem, for the rest of the paper we call this setting the 'two-class' setting. Denote by D_a the set of attacker sentences available for training. For each model M, let $t_M(k_\sigma) = \mu_M + k_\sigma \sigma_M$ where μ_M and σ_M are estimated as described above. The *accuracy* of the model M with respect to the decision threshold is given by the ratio of the number of correctly classified strings to the total number of strings,

$$A(M, k_\sigma) = \frac{|\{V(x, M) \mid v < t_M(k_\sigma),\, x \in D_u\}| + |\{a \mid V(x, M) \geq t_M(k_\sigma),\, x \in D_a\}|}{|D_u| + |D_a|}$$

Let ε be any limit on the desired accuracy ($0 \leq \varepsilon \leq 1$), The *robustness* $R_\varepsilon(M)$ of the model M is defined as

$$R_\varepsilon(M) = \int_{k>0 \;:A(M,k) \geq 1-\varepsilon} A(M, k) dk.$$

That is, the robustness is the area below the accuracy curve viewed as a function of threshold magnitude. Note that in practice the robustness can be rather accurately estimated using the average accuracy of the model over a number of suitable k_σ representatives. Figure 2 depicts the accuracy curves of various $\texttt{lz78}(s, m)$ models. The areas enclosed by these curves and the 90% asymptote are the ε-robustness values of these models (with $\varepsilon = 0.1$).

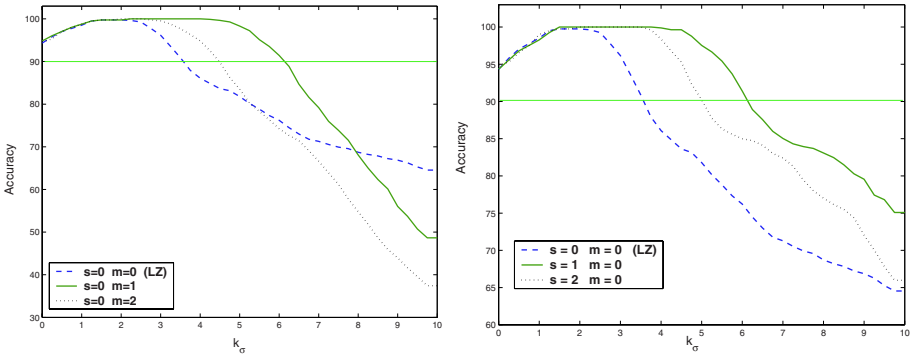

Fig. 2. Accuracy as a function of threshold magnitude for various $\texttt{lz78}(s, m)$ models. Areas above the 90% asymptote are robustness values. For example, $R_{0.1}(\texttt{lz78}(0, 1))$ and $R_{0.1}(\texttt{lz78}(1, 0))$ are the largest robustness values in the left and right panels, respectively.

The model M_u^* which maximizes robustness is selected in this two-class setting and k_σ is set to maximize accuracy as measured over the training set $D_u \cup D_a$. That is, $k_\sigma = \text{argmax}_k\, A(M_u^*, k)$. Note that there may be more than a single value which gives the maximum accuracy. In this case, there may be several peaks in the accuracy curve. We note however that in practice a single broad peak is typically observed. Whenever there is more than one maximum, we heuristically choose the threshold as the midpoint of the widest peak.

4 Dataset and Experimental Setup

For evaluating the proposed algorithms a dataset of keyboard event sentences was collected from 5 users and 30 attackers. We note that the recording of keyboard events including their precise time stamps is not straightforward using user-level programs on most standard operating systems. Thus, a suitably adapted system was constructed including a modified keyboard interrupt service routine[2]. Each of the users and attackers typed several sentences. The user input sequences were on average longer than the attackers' input. The text typed by

[2] In particular, a Linux system was used with all non-essential modules and services removed or disabled. System calls were used to request **unbuffered** keyboard events.

users corresponded to answers to open ended questions (e.g."What did you do today?") and to a specific sentence ("To be or not to be. That is the question."). Additionally a (completely) free text section was also allowed. On average, each user recorded 2551 ± 1866 keystrokes. Each of the thirty "attackers" was asked two open ended questions, and was required to type the specific sentence "To be or not to be. That is the question." They were also allowed to type in free text. On average 660 ± 597 keystrokes were logged for an attacker[3]. To maximize the utility of this dataset, the sentences, both before learning and before testing, were split into segments of 100 keystrokes (arbitrarily set). Additionally, all of the attackers' sentences (120 in total) were used to attack each model selected.

We selected a set of "feasible" parameter values for the models[4]. To maximize evaluation accuracy we used the following leave-one-out protocol: For each user u, each of the sentences in D_u was in turn selected to be in the test set and the rest of the sentences remained in the training set. Once a model was selected, it was tested if the model can identify and accept the left out sentence.

For the two-class problem, where we wish to see if providing attacker data can improve performance, the attackers were partitioned into two groups: a group of 10 attackers to be used for training (40 sentences), and a group of 20 attackers for testing (80 sentences). Other than the partitioning of the attackers, testing was identical to that of the single-class case, although the 10 attackers used for training were not used for testing. One hundred cross-validation folds were made.

5 Experimental Results

We begin by considering the results obtained for the single-class setting. Table 1 specifies the results in the single-class setting. As can be seen, impressive performance can be achieved by the system. The system performs well even when limited information is available (for example, user 5), though performance, particularly in self identification, does slightly suffer. Table 2 shows the results for the two-class experiments. Performance, on average, was similar to the single-class results, though user 5 did have a marked decline in self identification success. This does not seem to be dependent on the amount of data available, as user 1's performance also dropped, though less significantly. Performance in terms of successfully defending did improve, however, achieving perfect scores for nearly all of the users, which resulted in a higher break-even point.

In addition, we examined the performance of the algorithm when models were restricted to use the "pure" `lz78` algorithm (i.e. the `lz78`$(0, 0)$ model with Q and k_σ still variable was trained with the same methodology), both for the single-class and two-class problems. Due to space limitations we only report on the estimated break-even points for these experiments which were 93.57 and 96.42 for the single-class and two-class problems, respectively. These results indicate

[3] The complete dataset will be available at `http://www.cs.technion.ac.il/~rani/typist`.

[4] The particular values we tested are Q = 80, 90, 100, 110, 120 ; m = 0, 1, 2, 3, 4; s = 0, 1, 2, 3, 4; and k_σ = 0, 0.25, 0.5,0.75,. . . ,10.

Table 1. Single-Class Results: Individual users, averages and estimated break-even point (defined to be the harmonic mean of the averages).

User	# Keystrokes	# Sentences	# Self "Attacks"	Self ID (%)	# True Attacks	Defense(%)
1	5344	13	114	97.37 ± 2.79	1560	98.33 ± 1.08
2	4156	16	90	97.78 ± 8.53	1920	100.0 ± 0.63
3	1630	5	36	94.44 ± 11.65	600	99.67 ± 0.41
4	1076	5	23	91.3 ± 10.85	600	99.33 ± 0.97
5	548	5	14	92.86 ± 9.58	600	97.0 ± 3.82
Averages				**94.75 ± 2.51**		**98.87 ± 1.09**
Estimated Break-Even Point				**96.77**		

Table 2. Two-Class Results: Individual user, averages and break-even point. Results are across all 100 cross-validation folds.

User	# Keystrokes	# Sentences	# Self "Attacks"	Self ID (%)	# True Attacks	Defense(%)
1	5344	13	11400	93.86 ± 7.01	104000	100.0 ± 0.0
2	4156	16	9000	100.00 ± 0.0	128000	100.0 ± 0.0
3	1630	5	3600	97.22 ± 11.45	40000	100.0 ± 0.0
4	1076	5	2300	95.65 ± 11.23	40000	99.75 ± 0.5
5	548	5	1400	85.71 ± 17.2	40000	100.0 ± 0.0
Averages				**94.49 ± 4.83**		**99.95 ± 0.1**
Estimated Break-Even Point				**97.14**		

that the `lz78`(s, m) modifications have a significant advantage in the single-class setting, particularly when there is little data available for training. For example, for users 4 and 5, the estimated break-even points for the "pure" `lz78` algorithm are 90.2 and 80.9, respectively. With our improvements the values obtained for these users are 95.1 and 94.9, respectively.

6 Related Work

There is quite extensive literature on "keystroke dynamics" by attempting to identify characterizing features in keystroke sequences. One of the earliest works is [11], which introduce the use of "digraph times" in this context. For each pair of keys typed, its *digraph time* is the interval between the pressing of the first key and the pressing of the second. Many other works later use this basic idea or its extensions to "trigraphs", etc. Due to space limitations we limit the discussion here to two of the most recent papers, which present the most impressive results to-date. The work presented in [2] uses a combination of digraph times and keystroke latencies to generate feature vectors. Factor analysis is then used to select discriminative features. Using a nearest neighbor approach together with clustering, the authors examine the classification success rate of a number of distance functions. On a dataset consisting of 63 users, the best results are obtained using a Bayesian distance function. The stated results are approximately 92%. These results were obtained over a dataset where all users typed fixed text selections from "a list of phrases". There was also a free text component in this study though results are not presented and are stated to be inferior. The recent results of [12] consider again identifying typists of a fixed phrase. This phrase

consists of 683 characters (which form 125 words). Using a fixed trigraph vector representation the authors obtain very high accuracy using a heuristic distance measure between trigraph vectors. Their best results for the single-class problem are 1.8% false alarm rate and 0.01% for "imposter pass" rate. The authors also test higher order "graphs" and experiment with subsets of the fixed phrase. While higher order graphs (e.g. 6-graphs) do not improve results, the use of sub-phrases can drastically increase the false alarm rate (e.g. by taking 1/4 of the phrase the false alarm rate increases to more than 12%). While these two works indicate that very high precision can be obtained in recognizing keystroke "signatures" over a fixed text, these methods fall short in handling free text, particularly when little data is available. The main contribution of the present work is in showing for the first time a new representation and algorithms that can attain very high accuracy also for ***free text***. The results we obtain (e.g. over 96% break-even for the single-class authentication problem) enable a practical behaviometric security system for continual non-intrusive authentication, which can handle any text. These results are not directly comparable to the above results. However, when considering sample sizes and accuracy, it appears that our results may be significantly better than the results of [12]. Nevertheless, these other results are obtained with an impressive database consisting of typed sentences from 44 users and 110 attackers whereas our primarily free text dataset consists of 5 users and 30 attackers.

As noted previously, the more challenging and perhaps common setting for a security system as described here, is that of a single-class problem. This variant of binary classification has various other jargon names, such as: novelty detection outlier detection, one-class classification. For other approaches for setting the boundary in single-class problems see e.g. [13–15].

7 Conclusions and Future Work

We have introduced an approach to modeling keystroke dynamics of users based on using the universal Lempel-Ziv compression algorithm as a generator of the predictive distribution of future strings, based on statistics collected from an individual user. We use this predictive distribution in the context of single-class learning, where particular values for the augmented Lempel-Ziv algorithm are selected based on cross-validation. While previous work tended to focus on fixed representations based on N-graphs which can be considered to be fixed order Markov models, our representation allows for variable length contextual information. As a result of this, our statistical model is capable of retaining more robust statistics, possibly at the cost of increased space requirements.

Our method can be potentially improved in several ways. First, other universal prediction algorithms (such as CTW; see Section 3.2) could perhaps improve prediction accuracy, at the expense of speed. Such a compromise may be unacceptable for the particular application of continual non-intrusive authentication.

It may be interesting to investigate whether taking *relative* time-differentials (rather than absolute time-differentials) can improve performance, perhaps by a

reduction of the variance caused by the variability in typing speeds of users. This direction is particularly promising when considering the successful technique of [12], which achieved impressive performance on a fixed text by ignoring absolute differential times (but utilizing the relative sizes of trigraph times).

While our results are impressive, they can only be viewed as a proof-of-concept due to the limited sample size, and the use of a single session for data acquisition. Finally, an advantage of our techniques is that they are not specifically targeted to the keyboard, and can be easily extended to other devices.

References

1. V. Matyas Jr. and Z. Riha. Biometric authentication systems. Technical report, ECOM-MONITOR, 2000.
2. F. Monrose and A.D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.
3. A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, 1992.
4. F. M. J. Willems, Y. M. Shtarkov, and Tj. J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Trans. Info. Theory*, pages 653–664, 1995.
5. G. Manzini. The burrows-wheeler transform: Theory and practice. In *Symposium on Mathematical Foundations of Computer Science (MFCS '99)*, volume 1672, pages 34–47. Springer Verlag Lecture Notes in Computer Science, 1999.
6. J. G. Cleary and W. J. Teahan. Unbounded length contexts for PPM. *The Computer Journal*, 40(2/3):67–75, 1997.
7. J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.
8. G.G. Langdon. A note on the lempel-ziv model for compressing individual sequences. *IEEE Transactions on Information Theory*, 29:284–287, 1983.
9. M. Feder. Gambling using a finite state machine. *IEEE Transactions on Information Theory*, 37:1459–1465, 1991.
10. D.R. Stirzaker G.R. Grimmett. *Probability and Random Processes*. Oxford University Press, third edition, 2002.
11. R. Gaines, W. Lisowski, , S. Press, and W. Shapiro. Authentication by keystroke timing: Some preliminary results. Report R-256-NSF, Rand Corp., 1980.
12. F. Bergadano, D. Gunetti, and C. Picardi. User authentification through keystroke dynamics. *ACM Transactions on Information and System Security*, 5(4):367–397, 2002.
13. C. Bishop. Novelty detection and neural network validation. *IEEE Proceedings on Vision, Image and Signal Processing*, 141(4):217–222, 1994.
14. N. Japkowicz. *Concept-Learning in the absence of counterexamples: an autoassociation-based approach to classification*. PhD thesis, Rutgers, New Brunswick, 1999.
15. D.M.J. Tax. *One-Class Classification*. PhD thesis, The Delft University of Technology, 2001.