Typing Biometrics for User Authentication - a One-shot Approach

Josef Malmström
josefmal@stanford.edu
CS 229 - Stanford University

Hannes Lindström
halindst@stanford.edu
CS 230 - Stanford University

Abstract—**In a world of bad passwords, alternative methods for user authentication is becoming increasingly important. Keystroke dynamics is the time series data describing the keyboard typing rhythm of a person. It has been shown that this data can be used as a biometric for identity authentication [1]. While a variety of different methods to perform such authentication has been presented, most pose significant challenges in generalizing to users for which the number of available typing samples is small. In this paper we propose a novel method, similar to that of FaceNet [2], where we use metric learning to learn an embedding of the high dimensional typing samples into a lower dimensional space, in which the task of performing comparisons between samples is greatly simplified. We are able to show that our approach reaches a False Acceptance Rate (FAR) of 10.14 % and a False Rejection Rate (FRR) of 15.26 % on samples from users that were never seen during training.**

I. Introduction

In the past decade there has been significant developments in using biometrics as a means for user authentication. Biometrics that can be used for authentication come in many forms, that can broadly be separated into two categories: physical (e.g. fingerprints, face recognition) and behavioral (e.g. typing rhythm, voice, gait). In this project we specifically explore ways of using typing rhythm to authenticate users.

Keystroke dynamics is the time series data describing when and which keys are pressed and released as someone is typing on a keyboard. The primary motivation behind the study of such data is that it has been proved to be an effective unique identifier of a person, and can therefore be effectively used as a means of authentication [1]. In particular, using keystroke dynamics has many advantages compared to other authentication methods. The keystroke dynamics of a user can be recorded through a regular keyboard, meaning no specialized hardware is required. It has also been shown that keystroke dynamics can be used to continuously authenticate a user of by tracking their typing and rejecting them if the typing rhythm suddenly changes [3].

II. Related work

Historically, researchers have experimented with a plethora of approaches to using keystroke dynamics for the purpose of authentication. An overview of the more popular methods is provided by [1]. Broadly, most methods can be categorized as either purely statistical, based in machine learning, or in deep learning. Statistical methods range in complexity, from using generic statistical measures to form a notion of distance between typing samples, to more advanced probabilistic modeling (e.g. using Hidden Markov Models). A range of different machine learning techniques have proved to be somewhat effective at differentiating users based on their typing data. Methods that, under the most favorable circumstances, have been shown to reach False Acceptance Rates (FARs) and False Rejection Rates (FRRs) of below 10 % include for instance usage of SVM, random forest decision trees, Gaussian Mixture Models (GMM), K-means and Naive Bayes [1].

Most recently, there have been multiple attempts at applying deep learning to the problem of keystroke authentication, for instance by [3] and [4], which have mostly proven to reach better or equal performance compared to traditional methods. A significant challenge with deep learning approaches however, is the need for large amounts of data on a per user basis. If for instance a deep neural network is applied as a multi-class classifier to identify users, the network needs to be trained on significant amounts of data from each user. Furthermore, if a new user needs to be enrolled into the system we would not only need new data, but the architecture would need to be modified and retrained. Similar challenges occur if training a One versus Rest (OvR) style classifier for each enrolled user.

Key	Event	Time stamp
O	KeyDown	63578429797173
R	KeyDown	63578429797235
O	KeyUp	63578429797313

TABLE I: Structure of the raw typing data.

III. Dataset and features

The dataset used for this project is from a 2016 study on keystroke dynamics [5]. The dataset consists of text files of raw typing data from 148 different users. The data is formatted as presented in Table I[1].

As a feature representation of the raw data, we choose to look at the sequence of digraphs (i.e. key sequences of length two) and a set of attributes associated with each digraph. Specifically, we represent each digraph as a vector

$$\phi = \begin{bmatrix} \text{KD} & \text{H}_1 & \text{H}_2 & \text{PP} & \text{RP} \end{bmatrix} \quad \phi \in \mathbb{R}^5$$

where KD is an approximation of Manhattan distance between the two keys on the keyboard, H_1 and H_2 are the hold times for

[1] Note that the data in this table is mock data just to show the structure of the dataset.

the first and second key respectively (i.e the time elapsed from pressing to releasing the key), PP is the press-to-press time for the two keys (i.e. the time elapsed from pressing the first key to pressing the second key), and RP is the release-to-press time (i.e. the time elapsed from releasing the first key to pressing the second key).

While there are many other ways to represent the typing data for machine learning purposes (such as trigraphs or n-graphs), the digraph representation is the most common one, according to [1] among others, which is why we choose this representation. It is also worth noting that a fairly common practice is to only include a subset of the hold, press-to-press and release-to-press times as features. Here, we choose the most exhaustive set of features with the motivation that the deep learning approaches we will apply can learn which features are most critical.

After extracting this feature representation from the raw data we also segment the resulting sequences of digraphs into samples of length m, where we experiment with different values of m approximately in the range $[15, 100]$. Each segment of digraphs $x \in \mathbb{R}^{m\,x\,5}$ represents one data sample to be used for training (or evaluating) a model. A corresponding one-hot label $y \in \mathbb{R}^{K}$ indicating which of the K users in the dataset this sample belongs to is also generated.

IV. Baseline models

A. Baseline using GMMs

To get an appropriate baseline on our specific data set, we have implemented an approach described by [6], that uses Gaussian Mixture Models (GMMs). To give a brief overview of the method, it uses the EM algorithm to fit a GMM to the set of press-to-press times associated with the digraphs of one particular pair of keys. After a GMM has been fit to each the digraphs of each pair of keys, we have a typing profile for each user, represented as a set of Gaussian mixtures (one for each key-pair). To evaluate a new typing sample claimed to belong to a particular user, we run an algorithm, described in detail by [6], that computes a score in relation to the claimed user's typing profile. A high score indicates that it is likely that the provided typing sample did in fact come from the claimed user, while a low score indicates the contrary. Using validation data, an appropriate threshold for the score is chosen. If the sample achieves a score equal to or higher than the threshold, it is accepted, otherwise it is rejected.

Using this approach on our dataset we are able to reach a FAR of 14.6 % and a FRR of 6.7 %.

B. Baseline using CNN classifiers

One approach to user authentication with keystroke dynamics, for example utilized by [7], is to train an OvR classifier for every enrolled user to recognize whether a provided typing sample is from the user in question ($y = 1$) or not ($y = 0$). We have implemented a version of this approach, with a slightly modified architecture compared to [7]. The architecture we used is shown in Figure 1. For all convolutional layers a kernel of size 2 and a stride of 1 was used. The ReLU activation function was used for all hidden layers.

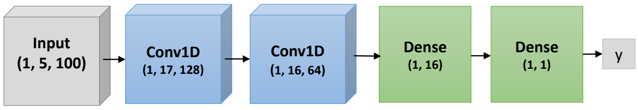


Fig. 1: The architecture used as a classifier for the typing data of a single user.

Training this architecture on the logistic loss of 90 % of the typing data from 10 different users, validating on 5 % and testing on the remaining 5 %, we were able to achieve the FARs and FRRs presented in Table II.

User	0	1	2	3	4	5	6	7	8	9
FAR	0.17	0.08	0.09	0.19	0.23	0.001	0.016	0.25	0.14	0.02
FRR	0.08	0.09	0.06	0.06	0.11	0.23	0.18	0.03	0.08	0.05

TABLE II: Results for per-user CNN classifiers on 10 random users.

We can see that both the FAR and FRR for many of the users are acceptable compared to other approaches, but that the results are fairly inconsistent across different users. Considering that the network was trained in the exact same configuration and with the same hyperparameters, results may however improve if this was tuned to each user. Even so, the challenges in employing this approach in an actual authentication system would be considerable, as described in a previous section.

V. Methods

In this section we provide the details of our metric learning based approach. Much like in FaceNet [2], we create a model that learns an embedding of the high dimensional typing samples into a lower dimensional space. The goal is to create embeddings such that embedded samples from the same user are close, and embedded samples from different users are distant. This is achieved by training the network using a triplet loss function. After training the embedding network, a second model is trained in order to perform the predictions. This second model is a SVM which is tasked with deciding whether embedded samples are from the same user or not. Below, we describe these two elements of the approach in greater detail.

A. Embedding network

Triplet loss: From the typing samples we form triplets, each consisting of an anchor A, a positive P and a negative N. The anchor is an arbitrary reference sample, the positive is a sample from the same user as the anchor, and the negative is a sample from a different user than the anchor. The triplet loss with respect to a single triplet is defined as

$$l(A, P, N) = \max(||A_e - P_e||_2 - ||A_e - N_e||_2 + \alpha, 0) \quad (1)$$

where A_e, P_e and N_e are the embeddings of the triplet, as computed by the embedding network, and α is a hyperparameter referred to as the margin. It is clear that this loss function encourages A_e and P_e to be close and A_e and N_e to be distant, which is what we want. We also see that the margin α determines how much larger than the AN-distance the corresponding AP-distance needs to be for us to be fully satisfied and yield a zero loss. Given a batch of n triplets, our objective function is defined simply as $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} l(A_i, P_i, N_i)$.

Online triplet mining: A critical problem in triplet learning is that for randomly formed triplets, many triplets will already yield a zero loss because $||A_e - P_e||_2 + \alpha \leq ||A_e - N_e||_2$. If many such triplets are used to train the embedding network, training will be very slow and may not converge at all since most triplets provide no useful information to the network. A method introduced by [2] involves selectively generating each batch of triplets online, as the network is being trained. Specifically, they present a variety of approaches employing the selection of so called *hard* triplets that satisfy the constraint $||A_e - P_e||_2 - ||A_e - N_e||_2 \geq 0$, and/or *semi-hard* triplets that satisfy the constraint $-\alpha \leq ||A_e - P_e||_2 - ||A_e - N_e||_2 < 0$.

For instance, [2] present a method referred to as *batch-all* where, for each mini-batch of data they generate all valid triplets but keep only triplets that are hard or semi-hard. We use a slightly modified version of this approach, where we generate all valid triplets for the mini-batch, but only keep the semi-hard triplets.

Architecture: The architecture of our embedding network is illustrated in Figure 3. The network takes typing samples in $\mathbb{R}^{5\times100}$ (i.e. $m = 100$) and generates embeddings in $\mathbb{R}^{40}$. The convolutional layers of the network consist of an Inception-like [8] architecture. All convolutional and fully connected layers in the network use the ReLU activation function. For the convolutional and max-pooling layers, we use ’same’ padding.

The reasoning behind using a shallow embedding network can be described by a few observations about our data. First of all, the data does not contain very many latent features. Most computer vision tasks require the network to be able to recognize very complex patterns spanning large parts of the input (which itself usually is much larger). In the task we are facing, however, the patterns we are attempting to identify can be assumed to be fairly local. The relationship between how the keys are pressed down at the start of a sentence in comparison to the end of another sentence two minutes later, is not very important. It might be more interesting to localize these short-term patterns, and then average them over the entire input. This does of course remove some of the expressive abilities of the network, however, it can be an effective method for combating overfitting. This is our motivation behind the usage of a global average pooling layer in the network, which also decreases the number of parameters in the dense layer thereby further decreasing the risk of overfitting. In comparison to other models that were tested, this combination of the Inception-style layer and global average pooling worked the best in allowing the network to create fairly expressive embeddings while not overfitting the training data.

In addition to the convolutions, two important aspects of the embedding network is the Batch- and L2-normalization layers. This combination allowed for significantly higher learning rates without having the embeddings collapse into a single point. These results can mainly be attributed to the Batch-Normalization layer, which is known to be able to reduce the number of training epochs required [9]. However, the two types of normalization were found to work best when used in combination. Another advantage of L2 normalizing the embeddings is that since all embeddings are contained on the unit hypersphere, it is possible to use a constant margin in an effective manner.

Hyperparameters: There is a variety of hyperparameters to be set for the embedding network. We found the most critical parameters to be the learning rate used during training, as well as the triplet loss margin α. For the the learning rate, we used 0.5×10^{-2}. We experimented with a range of values for the triplet margin α in the interval $[0, 1]$. We found that a value of $\alpha = 0.3$ gave the best results. Generally we found larger mini-batch sizes to be beneficial for training, as this means there will be a larger number of valid semi-hard triplets in each batch. However, since all valid triplets in a mini-batch must be generated in order to identify the semi-hard ones, and the number of valid triplets grows cubically with the mini-batch size, the choice of size is strongly limited by the amount of available memory. For our machine, we found a mini-batch size of 128 to be a reasonable trade-off.

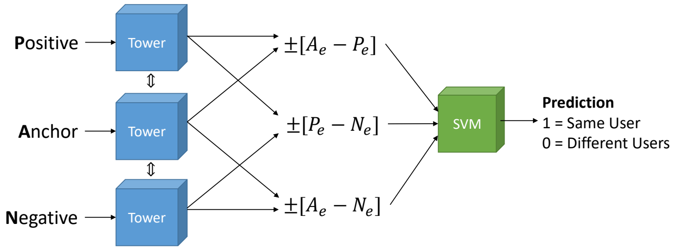


Fig. 2: Illustration of the prediction model.

B. Prediction model

Given a trained embedding model, we still need a model that can make predictions on a given pair of embedded samples to determine whether they are from the same user ($y = 1$) or different users ($y = 0$). To create such a model, we train a Support Vector Machine (SVM) on the difference between embeddings (element-wise). Much like in training the embedding network, we form triplets (A_e, P_e, N_e), and let the element wise differences between pairs in the triplet be training examples for the SVM model, with the appropriate label (i.e. 1 if the embedded samples are both from the same users, and 0 if the embedded samples are from different users). This procedure is illustrated in Figure 2. Note that we let both signs of the element wise differences be training examples, since for instance $[P_e - A_e]$ is an equally valid example as $[A_e - P_e]$.

To use the trained prediction model to make predictions on samples of data, we trial two different methodologies:

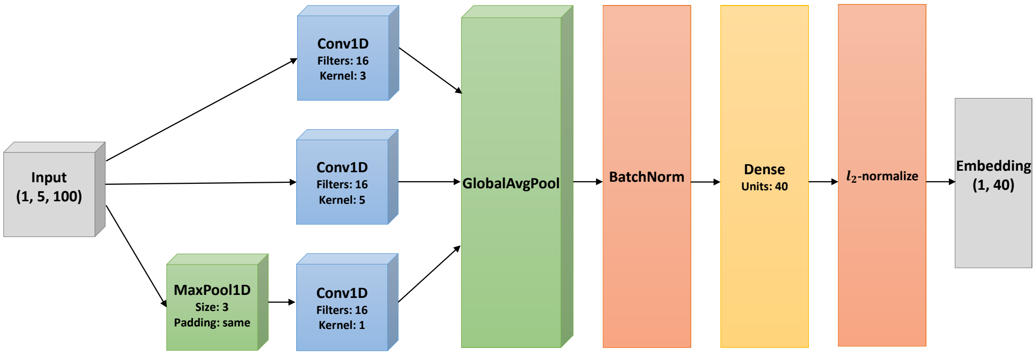


Fig. 3: The architecture used in the embedding network.

- The query sample is matched with a single reference sample from the same user. Both are fed through the embedding network, and their element wise difference are fed through the prediction model. The output of the SVM is used as prediction.
- The query sample is matched with five different reference samples from the same user. All are fed through the embedding network, and the element wise difference of each query-reference pair are fed through the prediction model. The majority vote of the outputs from the SVM for each query-reference pair is used as prediction.

Hyperparameters: To tune the hyperparameters of the SVM model, we performed a grid search on validation data across different kernels (linear, polynomial and RBF), and an exponentially growing range of values for C and γ. We found an RBF kernel, $C = 1$ and $\gamma = 1$ to yield the best results.

VI. Results

We trained our embedding network and prediction model on 90 % of the typing samples from 30 different random users, validated on 5 % and tested on the remaining 5 %. We also tested our trained system on all the typing samples from 30 other random users, that were not seen during training.

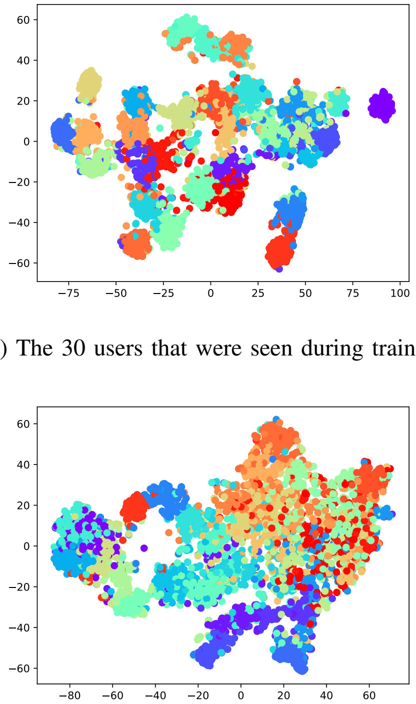
Resulting FAR and FRR when testing with the two prediction methodologies on samples from the 30 users that were seen during training and the 30 users that were not seen during training can be found in Table III and IV respectively. As a means for visualizing the learned embedding, we run the t-SNE dimensionality reduction algorithm on the training data as well as on samples from the 30 users that were not seen during training. Resulting t-SNE plots can be found in Figure 4.

	single reference	5 references
FAR	8.69 %	7.63 %
FRR	12.29 %	6.61 %

TABLE III: Non-generalized results.

	single reference	5 references
FAR	12.05 %	10.14 %
FRR	19.75 %	15.26 %

TABLE IV: Generalized results.



(a) The 30 users that were seen during training.

(b) 30 random users that were not seen during training.

Fig. 4: t-SNE of embeddings for samples from users that were seen during training, and other random users. Each color represents a different user.

VII. Discussion

Comparing the results to the performance of our baselines as well as a variety of other methods, from [1] for instance, we see that our approach is on par with other methods in terms of FAR and FRR. In addition, we see that by using the 5 reference majority vote prediction methodology, we can reach a respectable performance on users that were not seen during training, with a FAR of about 10 % and a FRR of about 15 %.

It is also worth noting that these results were attained by training on the data from only 30 different random users.

One could expect that the generalizability of the model would improve if it was trained on data from a wider variety of users. However as the number of users increase, training the embedding network also becomes more challenging.

For the embedding network, we experimented with a variety of different architectures and found early on that convolutional architectures seemed to yield better performance than recurrent models. However, our experiments with recurrent architectures were fairly limited and it could therefore be of interest in future work to explore further whether a recurrent model can be effectively used as an embedding network in this application.

Another noteworthy detail is that all the keystroke dynamics data we used, for training as well as testing, was generated using the same model of keyboard. It is expected that generalizing to different keyboard models is more challenging, as a given person typically has a different typing rhythm on different keyboards. The typing patterns of a person can also be highly variable due to a range of different factors other than the choice of keyboard. Studies have for instance shown that a person's typing rhythm changes significantly when they are tired [10]. One can also imagine that someone's typing rhythm is drastically different when typing with just one hand instead of two. In general, further studies are needed to explore the effect of these factors on an authentication system, and to potentially come up with methods for maneuvering them. For example, a possible approach for handling a variety of these factors could be to detect and maintain multiple identities for each user, based for instance on which keyboard they are using, their physical or mental state etc.

On the other hand, there could also be possibilities to leverage the variability caused by these factors for other applications. For example, one could experiment with using keystroke dynamics to detect fatigue or intoxication in workplaces where prolonged focus is critical (e.g. emergency phone services, flight control).

VIII. Conclusion and Future work

The proposed metric learning approach worked well on the studied test cases, reaching a FAR of 7.63 % and a FRR of 6.61 % on samples from users that were seen during training, and a FAR of 10.14 % and a FRR of 15.26 % on users that were not seen during training. Future aspects to be studied include how this system functions at scale, when the number of users is in the hundreds or thousands and whether this approach can be extended to work well for users that switch between different keyboard models. Additionally, an interesting aspect would be to study whether this type of system could be effectively used for other applications, such as detecting fatigue or intoxication.

IX. Code

The source code for the project is available at https://github.com/lm-strom/typing-net.

X. Contributions

This was a joint project between two courses: CS 230 (Hannes) and CS 229 (Josef). Since the project is not clearly separable into the parts done for CS 229 and the parts done for CS 230 without loss of context, we have submitted the same report for both classes. Both group members collaborated closely on nearly all parts of the project, and it is therefore difficult to distinctly separate which parts were worked on by who. While we believe such a split does not really make sense, a somewhat arbitrary separation is given below. Any part of the project not listed below was undoubtedly done in equal part by both group members.

Josef (CS 229)

- Implemented GMM baseline.
- Implemented OvR classifier baseline.
- Implemented online mining of triplets.
- Implemented, tuned and trained the SVM prediction model.

Hannes (CS 230)

- Implemented preprocessing of raw typing data into the digraph feature representation.
- Implemented, tuned and trained the embedding network.

References

[1] P. S. Teh, A. B. J. Teoh, and S. Yue, "A survey of keystroke dynamics biometrics." *TheScientificWorldJournal*, vol. 2013, p. 408280, 11 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24298216http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3835878

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," pp. 815–823, 2015. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Schroff_FaceNet_A_Unified_2015_CVPR_paper.html

[3] L. Xiaofeng, Z. Shengfei, and Y. Shengwei, "Continuous authentication by free-text keystroke based on CNN plus RNN," *Procedia Computer Science*, vol. 147, pp. 314–318, 1 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919302935

[4] H. Ceker and S. Upadhyaya, "Sensitivity analysis in keystroke dynamics using convolutional neural networks," in *2017 IEEE Workshop on Information Forensics and Security (WIFS)*. IEEE, 12 2017, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8267667/

[5] Y. Sun, H. Ceker, and S. Upadhyaya, "Shared Keystroke Dataset for Continuous Authentication," in *8th IEEE International Workshop on Information Forensics and Security*, Abu Dhabi, UAE, 2016. [Online]. Available: https://cubs.buffalo.edu/research/datasets

[6] H. Ceker and S. Upadhyaya, "Enhanced recognition of keystroke dynamics using Gaussian mixture models," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*. IEEE, 10 2015, pp. 1305–1310. [Online]. Available: http://ieeexplore.ieee.org/document/7357625/

[7] P. Kobojek and K. Saeed, "Application of Recurrent Neural Networks for User Verification based on Keystroke Dynamics," *Journal of Telecommunications and Information Technology*, no. nr 3, pp. 80–90, 2016. [Online]. Available: https://www.infona.pl/resource/bwmeta1.element.baztech-c0824cfa-8493-417d-8caa-6400754aca32

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," Tech. Rep. [Online]. Available: https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

[9] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," pp. 448–456, 2015. [Online]. Available: https://dl.acm.org/citation.cfm?id=3045167

[10] M. Ulinskas, M. Woźniak, and R. Damaševičius, "Analysis of Keystroke Dynamics for Fatigue Recognition." Springer, Cham, 2017, pp. 235–247. [Online]. Available: http://link.springer.com/10.1007/978-3-319-62404-4_18