Mining Keystroke Timing Pattern for User Authentication

Saket Maheshwary(✉) and Vikram Pudi

Center for Data Engineering and Kohli Center on Intelligent Systems,
International Institute of Information Technology-Hyderabad, Hyderabad, India
saket.maheshwary@research.iiit.ac.in, vikram@iiit.ac.in

Abstract. In this paper we investigate the problem of user authentication based on keystroke timing pattern. We propose a simple, robust and non parameterized nearest neighbor regression based feature ranking algorithm for anomaly detection. Our approach successfully handle drawbacks like outlier detection, scale variation and prevents overfitting. Apart from using existing keystroke timing features from the dataset like dwell time and flight time, other features namely bigram time and inversion ratio time are engineered as well. The efficiency and effectiveness of our method is demonstrated through extensive comparisons with other state-of-the-art techniques using CMU keystroke dynamics benchmark dataset and has shown great results in terms of average equal error rate (EER) than other proposed techniques. We achieved an average equal error rate of **0.051** for the user authentication task.

Keywords: Anomaly detection · Feature ranking · Nearest neighbor · Regression · Prediction · Security

1 Introduction

In this era where everyone wants secure, faster, reliable and easy to use means of communication, there are many instances where user information such as personal details and passwords get compromised thus posing a threat to system security. In order to tackle the challenges posed on the system security biometrics [8] prove to be a vital asset. Biometric systems are divided into two classes namely physiology based ones and the ones based on behavior. Physiology based approach allows authentication via use of retina, voice and fingerprint touch. In contrast, behavior based approach includes keystroke dynamics on keyboard or touch screens and mouse click patterns.

In this paper we propose a learning model to deal with *keystroke dynamics* – a behavior based unique timing patterns in an individuals typing rhythm which is used as a protective measure. These rhythms and timing patterns of tapping are idiosyncratic [1] the same way as handwriting or signatures are, due to their similar governing neurophysiological mechanisms. Back in the 19th century, telegraph operators could recognize each other based on ones specific tapping style [15]. Based on the analysis of the keystroke timing patterns, it is possible to differentiate between actual user and an intruder. By keystroke dynamics we refer to any feature related to the keys that a user presses such as key down time, key up time, flight time etc. In this paper, we concentrate on classifying users based on static text such as user password. The mechanism of keystroke dynamics can be integrated easily into existing computer systems as it does not require any additional hardware like sensors thus making it a cost effective and user friendly technique for authenticating users with high accuracy. It is appropriate to use keystroke dynamics for user authentication as studies [21,22] have shown that users have unique typing patterns and style. Moreover [21,22] has proven some interesting results in their research work as well. First, [21,22] proved is that the users present significantly dissimilar typing patterns. Second they have shown details about the relationship between users occurrence of sequence of events and their typing style and ability. Then [21,22] explained sequence of key up and key down events on the actual set of keys. Then [21,22] have also shown that there is no correlation between users typing skills and the sequence of events. Hence all these factors make it difficult for intruders to match with the actual users typing patterns. Keystroke dynamics is concerned with users timing details of typing data and hence various features could be generated from these timing patterns. In this paper we are using timing features only on static text.

The rest of the paper is organized as follows. In Sect. 2 we discuss related work and our contribution. In Sect. 3 we discuss the details of how manual features are engineered from the dataset and in Sect. 4 we discuss the concept of optimal fitting line. In Sect. 5 we present our proposed algorithm for feature ranking which is divided into two sub sections where first subsection discusses proposed approach on how feature ranking is done using nearest neighbor regression and second subsection discusses the neural network we used on the ranked or weighted feature space for anomaly detection. In Sect. 6 we experimentally evaluate our algorithm and show the results. Finally, we conclude our study and identify future work in Sect. 7.

2 Related Work

Classifying users based on keystroke timing patterns has been in limelight when [6] first investigated whether users could be distinguished by the way they type on keyboard. Researchers have been studying the user typing patterns and behavior for identification. Then [7] investigated the possibility of using keystroke timings as to whether typists could be identified by analyzing keystroke times as they type long passages of text. Later [17] extracted keystroke features using the mean and variance of digraphs and trigraphs. A detailed survey [18] on the keystroke dynamics literature using the Euclidean distance metric with Bayesian like classifiers. Initially [3] and later [9] proposed to use the relative order of duration times for different n-graphs to extract keystroke features that was found to be more robust to the intra-class variations than absolute timing. Also [9] published great results for text-free keystroke dynamics identification where they merge relative and absolute timing information on features. Then [23] proposed a new distance metric by combining Mahalanobis and Manhattan distance metrics. Many machine learning techniques have been proposed as well for keystroke dynamics as an authentication system. Keystroke dynamics can be applied with variety of machine learning algorithms like Decision Trees, Support Vector Machines, Neural Networks, Nearest Neighbor Algorithms [5] and Ensemble Algorithms [19] among others.

One problem faced by researchers working on these type of problems is that majority of the researchers are preparing their own dataset by collecting data via different techniques and the performance criteria is not uniform as well hence comparison on similar grounds among the proposed algorithms becomes a difficult task. To address this issue, keystroke dynamics benchmark dataset is publicly provided with performance values of popular keystroke dynamics algorithms [12] to provide a standard universal experimental platform. They collected and published a keystroke dynamics benchmark dataset containing 51 subjects with 400 keystroke timing patterns collected for each subject. Besides this they also evaluated fourteen available keystroke dynamics algorithms on this dataset, including Neural Networks, KNNs, Outlier Elimination, SVMs etc. Various distance metrics including Euclidean distance, Manhattan distance and Mahalanobis distance were used. This keystroke timing pattern dataset along with the evaluation criteria and performance values stated provides a benchmark to compare the progresses of new proposed keystroke timing pattern algorithms on same grounds.

2.1 Our Contribution

The performance study of the fourteen existing keystroke dynamics algorithms implemented by [12] indicated that the top performers are the classifiers using scaled Manhattan distance and the nearest neighbor classifier. In this paper we present a new nearest neighbor regression based feature ranking algorithm for anomaly detection that assigns weight to the feature vector.

Nowadays neural network based models are frequently used in the field of computer vision, speech signal processing, text representation have now been adopted in the fields of security as well. These neural network based techniques have multiple advantages over previous approaches both in task specific performance and scalability. Motivated by the superior results obtained by the neural network models we decided to use it as a classifier for anomaly detection. We used a simple 3 layer neural network classifier for anomaly detection by giving the weighted feature space generated by our model as input to neural network. Our proposed approach has the following desirable features:

- **Parameterless:** We first design our nearest neighbor based regression algorithm and then show how the parameter can be automatically set, thereby resulting in a parameterless algorithm. This removes the burden from the user of having to set parameter values – a process that typically involves repeated trial-and-error for every application domain and dataset.

- **Accurate:** Our experimental study in Sect. 6 shows that our algorithm provides more accurate estimates than its competitors. We compare our approach with 14 other algorithm using the same evaluation criteria for objective comparison.
- **Robust/Outlier Resilient:** Another problem with the statistical approaches is outlier sensitivity. Outliers (extreme cases) can seriously bias the results by pulling or pushing the regression curve in a particular direction, leading to biased regression coefficients. Often, excluding just a single extreme case can yield a completely different set. The output of our algorithm for a particular input record is dependent only on its nearest neighbors hence insensitive to far-away outliers.
- **Simple:** The design of our algorithm is simple, as it is based on the nearest neighbor regression. This makes it easy to implement, maintain, embed and modify as the situation demands.

Apart from our proposed algorithm we have engineered two new features namely *Bigram time* and *Inversion Ratio time* as discussed in Sect. 3.

3 Feature Engineering

What are good timing features that classify a user correctly? This is still an open research problem. Though keystroke up, keystroke down and latency timing are the commonly used features, in this paper we have generated two new features from the given dataset besides the existing features. The dataset [12] provides three types of timing information namely the hold time, key down-key down time and key up-key down time. Besides these three existing features, two new features namely Bigram time and Inversion ratio time are engineered. Following are the details of five categories of timing features which is used to generate 51 features using keystroke timing dataset [12]. Figure 1 illustrates various timing features where up arrow indicates key press and down arrow indicates key release.

- **Hold Time** also known as dwell time, is the duration of time for which the key is held down i.e. the amount of time between pressing and releasing a single key. In Fig. 1, H_{i} represents the hold time.
- **Down-Down Time** key down key down time is the time from when key1 was pressed to when key2 was pressed. In Fig. 1, the times DD_{i} depicts the down time.
- **Up-Down Time** key up key down time is the time from when key1 was released to when key2 was pressed. This time can be negative as well. In Fig. 1, the times UD_i depicts the up down time.
- **Bigram Time** is the combined time taken by two adjacent keystrokes i.e. the time from pressing down of key1 to releasing to key2.
- **Inversion Ratio Time** it is the timing ratio of hold time of key1 and key2 where key1 and key2 are the two continuous keystrokes. In Fig. 1, $H_{\mathrm{i+1}}/H_{\mathrm{i}}$ is the inversion ratio time.

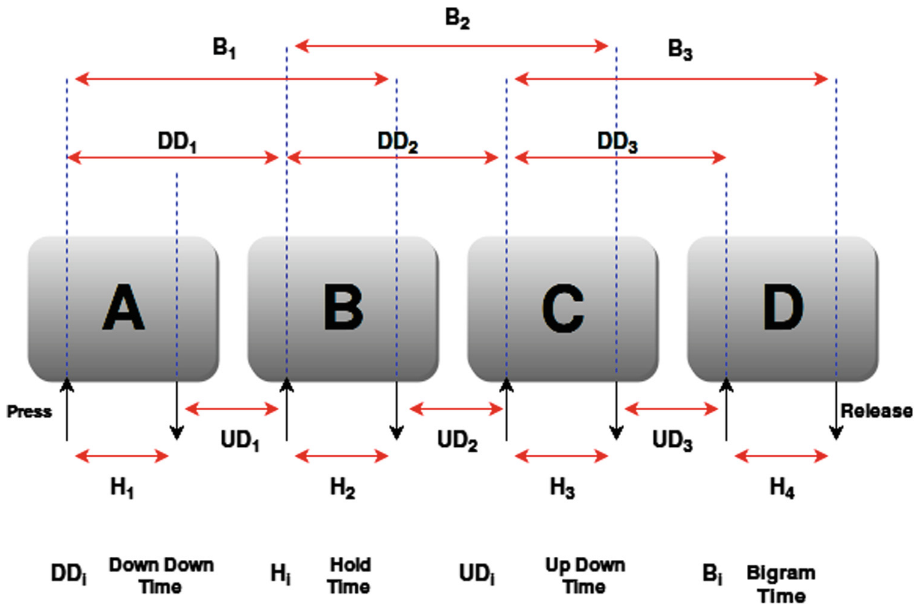


Fig. 1. Illustration of generated keystroke timing features where A, B, C, D are the keys

4 Optimal Fitting Line

Regression algorithms are used for predicting (time series data, forecasting), testing hypothesis, investigating relationship between variables etc. Here in this section we discuss how the optimal fitting line attempts to predict the relationship between one variable from one or more other variables by fitting a linear equation to observed data.

In this paper we assume that to construct the line of best fit, with increase or decrease in each independent variable value the dependent variable changes smoothly. Thus this helps us in achieving almost linear relationship between dependent and independent variables thus allowing us to optimally fit a line onto the points in a small neighborhood. The line which minimizes the mean squared error is referred to as optimal fitting line. A low value of error indicates that the line is optimally fitted to the neighborhood and has captured the linearity of the locality. Let the k points have values $\{(x_1, y_1), ..., (x_\mathrm{k}, y_\mathrm{k})\}$ in dimension x and y and let the variable to be predicted be y. Let the equation of line be of the form $y = ax + b$. Hence, dependent variable will take the value $ax_\mathrm{i} + b$ corresponding to tuple i. Let the error in prediction for tuple i be denoted as e_i and is equal to $|y - ax_\mathrm{i} - b|$. Hence the local mean squared error (LME) is denoted as,

$$LME(a, b) = \frac{\sum_{i=1}^{k} e_\mathrm{i}}{k} = \frac{\sum_{i=1}^{k} (y - ax_\mathrm{i} - b)^2}{k} \tag{1}$$

By minimizing LME where a and b are the parameters denoted by,

$$a = \frac{\sum_{j=1}^{k} y_{\mathrm{j}} \sum_{i=1}^{k} x_{\mathrm{i}} - k \sum_{i=1}^{k} x_{\mathrm{i}} y_{\mathrm{i}}}{\sum_{j=1}^{k} x_{\mathrm{j}} \sum_{i=1}^{k} x_{\mathrm{i}} - k \sum_{i=1}^{k} {x_{\mathrm{i}}}^{2}} \tag{2}$$

$$b = \frac{\sum_{i=1}^{k} y_{\mathrm{i}} - a \sum_{i=1}^{k} x_{\mathrm{i}}}{k} \tag{3}$$

Thus, we get the equation of the optimal fitting line. Now after constructing the line of best fit, we are able to predict the dependent values for test tuple. Then we compare the actual and the predicted values of dependent variable to calculate least mean error for the given test tuple. Now based on the mean error, we are assigning weights to our feature vector in inverse proportion which is discussed in Sect. 5.

5 Our Proposed Approach

5.1 Support for Categorical Attributes

In this section we discuss how our proposed approach deals with categorical data. The keystroke timing dataset that we used for evaluating our approach has categorical attributes. One of the serious limitations of existing regression algorithms is their support only for numeric attributes. So in order to tackle this problem we are using a similarity measure which helps us to quantify the relation between two classes using some real valued function. The section below explains the similarity function that we have used in this paper. Most of real life datasets have mixed attributes (set of both numeric and categorical type attributes) and hence to overcome this situation we are using *cosine similarity* measure. The dot product for two vectors $\boldsymbol{A} = (a_1, a_2, ...)$ and $\boldsymbol{B} = (b_1, b_2, ...)$ where a_{n} and b_{n} are the components of the vector and n is the dimension of the vector space. Hence the dot product between A and B is formulated as $\boldsymbol{A} \cdot \boldsymbol{B} = a_1 b_1 + a_2 b_2 + ... + a_{\mathrm{n}} b_{\mathrm{n}}$.

The cosine similarity between two vectors is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude, it can be seen as a comparison between timing vectors on a normalized space because we are not taking into consideration only the magnitude of each timing vector, but the angle between them. What we have to do to build the cosine similarity equation is to solve the equation of the dot product for the $\cos\theta$. Thus, the similarity values obtained using cosine similarity gives us a clear estimate of how similar the categorical values are with respect to the class labels.

$$\cos\theta = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{||\boldsymbol{A}||\,||\boldsymbol{B}||} \tag{4}$$

5.2 Proposed Algorithm

In this section we discuss our proposed nearest neighbor regression algorithm in detail. Our algorithm successfully eliminates nearest neighbor algorithm problems like choice of number of neighbors k by choosing the optimal k value corresponding to minimum error thus making our algorithm to be non parametric in nature. Our algorithm uses a unique weighing criteria (Algorithm 2) to assign weights to the feature vector hence enabling us to determine the relative importance of dimensions. The notation used for the algorithm is as follows: The training data has d dimensions with feature variables $(A_1, A_2, ..., A_\mathrm{d})$ and the value of the feature variable for the j^th feature variable A_j corresponding to the i^th tuple can be accessed as $data[i][j]$. The value of the dependent variable of the training tuple corresponding to id value i can be accessed as $y[i]$. The value of the dependent variable is calculated using the cosine similarity and k represents the number of nearest neighbors. For a given test tuple T the value of its k nearest neighbors is determined using an iterative procedure (line 4 of Algorithm 1) hence making our algorithm to be non parametric in nature. The range for value k is from *low* to *high* where *low* is set to value 5 (sufficiently small value) an *high* is set to *size of training data data/2* (sufficiently large value). Now we describe our algorithm using the pseudo code below shown in Algorithms 1 and 2.

We iterate for k in range *5* to *size of training data set/2* and calculate the k nearest neighbors for test data. The k evaluated neighbors are stored in list *ClosestNeighbors* (line 6 of Algorithm 1). Now Algorithm 2 constructs an optimal fitting line $Line_i$ for each dimension of our feature vector (the dataset used by us has 51 features) by fitting a linear equation to observed *ClosestNeighbors* list, in the plane of feature variable and the dependent variable. The regression line is constructed as discussed in Sect. 4. Using the parameters from the equation of the line a and b (Eqs. 2 and 3) we predict the dependent value of test data (line 4 of Algorithm 2). Based on the predicted and actual values of the dependent variable squared error E_i is calculated (line 5 of Algorithm 2).

Algorithm 1

```
 1: procedure KNN BASED DIMENSIONAL REGRESSION
 2: $MinimumError \leftarrow \infty$, $ErrorforK \leftarrow \infty$
 3: $OutputWeights \leftarrow 1$ // All $d$ dimensions have same weight initially
 4:     for each $k$= low to high do
 5:         $ErrorforK \leftarrow 0$
 6:         $ClosestNeighbors \leftarrow GetNeighbors(data, k, T)$
 7:         $DimensionalRegressor(T)$ // Algorithm 2
 8:         if $MinimumError > Errorfork$ then
 9:             $MinimumError \leftarrow Errorfork$
10:             $OutputWeight \leftarrow W_\mathrm{T}$
11:         end if
12:     end for
13: return $OutputWeight$
14: end procedure
```

Algorithm 2

```
1: procedure DIMENSIONAL REGRESSION
2:     for each i = 1 to d do // d is the number of dimensions
3:         $Line_i \leftarrow ConstructLine(ClosestNeighbors, i)$ // As discussed in Sect. 4
4:         $PredictedTestVal_i \leftarrow T_i * a + b$
5:         $E_i \leftarrow (PredictedTestVal_i - ActualTestVal_i)^2$
6:     end for
7:     if $\forall\ i\ E_i$ is equal then
8:         $W_T \leftarrow 1$
9:     else
10:        for each i = 1 to d do
11:            $weight_i \leftarrow max(E_i)/E_i$
12:            $W_T \leftarrow weight_i$
13:        end for
14:    end if
15: $Errorfork \leftarrow \sum_{j=1}^{d} E_j$
16: return $W_T$
17: end procedure
```

It would be appropriate to state that a lower error value in predicting the line indicates that the constructed regression line is optimal in nature and fits the neighborhood of test data. Hence we conclude that the value of dependent variable predicted via the line of best fit is approximately correct and thus a higher weight should be assigned for a more optimal line or we can say a line with lower squared error. This intuition is captured by assigning weights in inverse proportion to the error in prediction for this dimension, hence a feature with high error value is assigned lower weight and the feature with lower error value is assigned higher weight. The squared error in prediction of neighbors (line 15 of Algorithm 2) is computed and stored in $Errorfork$. A lower value of the squared error indicates that the weight values chosen using the nearest neighbors are appropriate. We then select the value of the parameter k for which the calculated error is minimum and hence assigns the corresponding weight vector W_T (line 8–10 of Algorithm 1). On this weighted feature vector we evaluate the anomaly score via a scaled Manhattan distance metric as discussed in the section below. The approach demonstrated in Algorithms 1 and 2 is a completely novel idea for dimension wise assigning weights in inverse proportion to error.

5.3 Neural Network for Anomaly Detection

After the weights have been assigned to the feature vector via our proposed algorithm, we calculate the anomaly scores as described by [12] for evaluating our model. For calculating anomaly score we are using a simple feed-forward neural network with a input layer with the size of our feature vectors and one hidden layer of 200 dimensions. After experimenting with different number of neurons in the hidden layer, we found out that results are best reported at 200 neurons. All the layers are fully connected. The higher size of hidden layer introduces sparsity in our network and helps in capturing the inter-feature relations which might be present. Following subsections explain other building blocks of neural network. We later discuss ablation studies for each in Table 2. We define the loss by the negative log-likelihood function which maximized the probability that sample gets classified as user or impostor. Learning is done through back-propagation of the losses through our network [10] (Fig. 2).

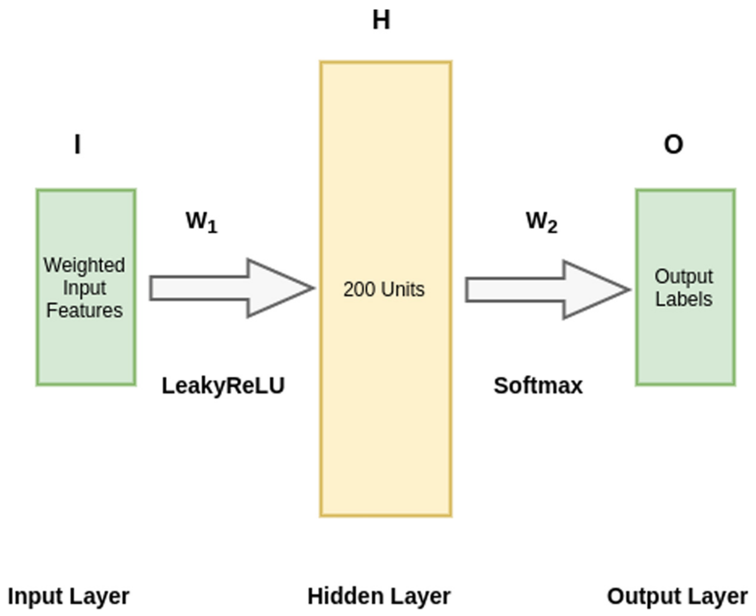


Fig. 2. Illustration of neural network architecture. Here **W1** and **W2** represents the weight matrices which are our parameters to be learnt. **H** is the hidden layers of size 200. **I** and **O** are input and output layers respectively.

5.4 Dropout

We use dropout [20] after our hidden layer which act as a regularizer and restricts over-fitting. During our training stage we randomly delete the nodes of each hidden layer with a certain probability p for each input sample. These neurons do not participate in the back-propagation learning. In testing time, the weights are correspondingly divided by $1-p$. Using dropout, forces the rest of the neurons in the hidden layers to learn more robust features and depend lesser on other specific neurons. In [20] more details are provided which show that using dropout can be an economic alternative to ensembling various network architectures.

5.5 Batch Normalization

After every fully-connected layer, we use batch normalization [11] before the respective activation functions. Using batch normalization we monitored the gap between training and testing loss over epochs narrowed down. This led to better generalization.

5.6 Leaky ReLU

Non-linear function Rectified linear unit (ReLU) is preferred to sigmoid or hyperbolic-tan because it simplifies backpropagation, makes learning faster while also avoiding saturation. However for large gradients, ReLU [16] can cause particular neurons to die and not participate in learning at-all. LeakyReLU's have a small positive gradient $f(z) = max(0.01x, x)$ which prevent this dying of a neuron. We applied Leaky ReLU as our activation function after the fully connected layers.

5.7 Adam

In recent times, several algorithms (with implemented software tools) are available for training a deep neural network. While stochastic gradient descent (SGD) for quite some time have been the top choice, there has been study which indicate some of the obvious flaws [14] in the vanilla implementation. There have been some attempts to automatically tune its learning rate thus resulting in much faster convergence. For anomaly detection we use Adam [13] instead of SGD which required a lot of fine-tuning with the learning rate and over 500 epochs to converge.

5.8 Inputs to Neural Network

The dataset consists of keystroke timing information of 51 users, where each user is made to type **.tie5Roanl** as password. All the 51 users enrolled for this data collection task typed the same password in 8 different sessions with 50 repetitions per session thus making each user to type 400 times in total (Table 1).

Table 1. Hyperparameters used in our neural network

Name	Specification
Dropout	0.3
LeakyReLU	0.01
Adam	0.001
Epochs	200

6 Experimental Setup

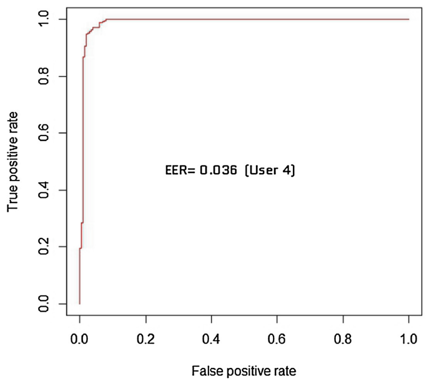
In this section we discuss the experimental setup, evaluation criteria used and the performance of our proposed model. We evaluated our approach on the CMU keystroke dynamics benchmark dataset [12] where *51* users were designated for this task. We demonstrate the effectiveness of our model with the average equal error rate (EER). We compare the results with various proposed anomaly detectors/classifiers which have been used in literature. We used the python library Keras [4] for building our neural network architecture. All our experiments were carried out on a Pentium 4^{th} generation machine with 4 GB memory. For experiments, we took the same 200 initial timing feature vector per-user as before. 10% of training data was kept aside as validation data for hyperparameter tuning.

6.1 Training

We frame keystroke dynamics based authentication as a one-class classification problem. For authentication, neural network learns one model per user, rejects anomalies to the learned model as impostors, and accepts inliers as the genuine user. Consider a scenario in which a users long-time password has been compromised by an impostor. We assume the user to be practiced in typing their own password, while the impostor is unfamiliar with it (e.g., typing it for the first time). We measure how well each of our detectors is able to discriminate between the impostors typing and the genuine users typing in this scenario. We start by designating one of our 51 subjects as the genuine user, and the rest as impostors. We train an anomaly detector by extracting 200 initial timing feature vectors for a genuine user from the dataset. We repeat this process, designating each of the other subjects as the genuine user in turn thus creating models equal to number of distinct subjects or users. Unlike most existing approaches, which only use actual users data at training time, our model leverage data from background users to enhance the models discriminative capability thus improving the prediction performance. We randomly took 5 samples from each background users as negative samples. Note that these 5 random samples were carefully chosen such that no impostor samples that were used in testing were shown during the time of training. For this problem setting, we use the evaluation criteria as mentioned in [12] in order to have comparison on same grounds.

6.2 Testing

We take last 200 passwords typed by the genuine user from the dataset. These 200 timing feature vectors acts as test data. Scores generated in this step acts as the user scores. Next, we take initial 5 passwords typed by each of the 50 impostors (i.e., all subjects other than the genuine user) from the dataset which acts as the impostor scores. Thus we form a test dataset of 200 positive samples and 250 negative samples per user which we provide to our neural network and record the output predictions. If s denotes the predictions, the corresponding anomaly score was calculated as $1 - s$ [12]. Intuitively, if s is close to 1.0, the test vector is similar to the training vectors, and with s close to 0.0, it is dissimilar (Fig. 3).



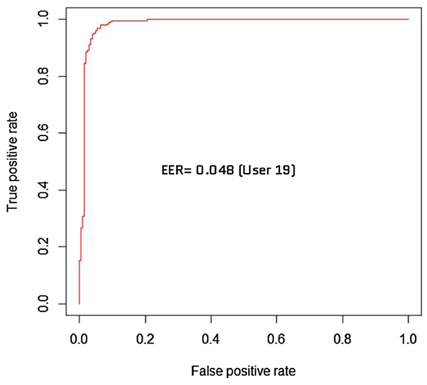


Fig. 3. Shows ROC curve for different users with their equal error rate (EER) value where user number corresponds to the user as labeled in CMU dataset.

6.3 Empirical Evaluation for User Authentication

Based on the genuine user scores and impostor scores generated in the steps above, we generate the ROC curve for the actual (genuine) user. Then we calculate the equal error rate from the ROC curve where the equal error rate corresponds to that point on the curve where the false positive rate (false-alarm) and false negative rate (miss) are equal. We repeat the above four steps, in total of 51 times where every time each of the subsequent user is taken as the genuine user from the 51 distinct users in turn, and calculate the equal-error rate for each of the genuine users. Finally we compute the mean of all 51 equal-error rates which gives us the performance value for all users, and the standard deviation which will give us the measure of its variance across subjects. In order to ensure comparison on same grounds we have used exactly the same evaluation criteria as stated by [12] on our proposed approach. The train-test data split was also kept the same.

Table 2. Ablation study of our neural network model showcasing the increase of performance for each additional component we use. Note that for this test we used tan-h activation when we opt not to use LeakyReLU. All components are added independently on the base model. Accuracy is based on the multiclass user recognition problem.

Model architecture	EER	Time taken (min)
Base model	0.0701	1.54
With Dropout	0.0674	3.18
With LReLU	0.0623	4.53
With Batch-Norm	0.0602	5.12
Our 3 layer model	**0.051**	5.81

7 Results

Table 3 shows the comparison of 16 other proposed keystroke timing algorithms with our proposed approach. Comparison is shown with 16 other algorithms which used the same dataset and the same evaluation criteria thus assuring an objective comparison. Our model is able to achieve an average equal error rate (EER) of ***0.051*** and with a standard deviation (stddev) of 0.042 across 51 subjects. The average equal error rate (EER) shown in the Table 3 are the fractional rates between 0.0 and 1.0, not the percentages. Clearly from Table 3, our model performs superior than other proposed techniques in comparison.

Table 3. Comparison of 16 different keystroke timing pattern algorithms that uses the same CMU keystroke timing dataset and evaluation criteria in terms of average equal error rate (EER) (with standard deviation shown in brackets).

Model/algorithm	Average EER (stddev)	Source
Our proposed algorithm (with 2 new engineered features)	**0.051(0.040)**	
Our proposed algorithm (without 2 new engineered features)	**0.054(0.042)**	
Median vector proximity	0.080(0.055)	[2]
Manhattan-Mahalanobis (no outlier)	0.084(0.056)	[23]
Manhattan-Mahalanobis (outlier)	0.087(0.060)	[23]
Manhattan (scaled)	0.0962(0.0694)	[12]
Nearest neighbor (Mahalanobis)	0.0996(0.0642)	[12]
Outlier count (z-score)	0.1022(0.0767)	[12]
SVM (one-class)	0.1025(0.0650)	[12]
Mahalanobis	0.1101(0.0645)	[12]
Manhattan (filter)	0.1360(0.0828)	[12]
Neural network (auto-assoc)	0.1614(0.0797)	[12]
Euclidean	0.1706(0.0952)	[12]
Fuzzy logic	0.2213(0.1051)	[12]
k Means	0.3722(0.1391)	[12]
Neural network (Standard)	0.8283(0.1483)	[12]

8 Conclusion and Future Work

In this paper we investigate the problem of authenticating users based on keystroke timing pattern. We engineered new features namely bigram time and inversion ratio time apart from the features already given in the CMU keystroke timing dataset. Besides engineering new features we also proposed a simple and robust nearest neighbor based regression algorithm. We evaluated our results and compared it against 14 other algorithms that used the same dataset and evaluation criteria thus providing performance comparison on equal grounds. Although simple, it proved to be effective as it outperformed competing algorithms as shown in Table 3.

Future work involves extending our work for soft keys or touch pad keys and in addition to timing pattern features we can use users pressure patterns as well in order to authenticate users. We are planning to experiment with different curve fitting techniques as well. We plan on extending our models to other available datasets on this domain. We would also like to investigate if transfer learning can help with user authentication and identification for large pool of users when trained from a limited dataset.

Acknowledgement. This work was supported by http://metabolomics.iiit.ac.in/ and we would like to thank them for their support.

References

1. Dvorak, A., Merrick, N., Dealey, W., Ford, G.: Typewriting behavior (1936)
2. Al-Jarrah, M.M.: An anomaly detector for keystroke dynamics based on medians vector proximity. J. Emerg. Trends Comput. Inf. Sci. **3**(6), 988–993 (2012)
3. Bergadano, F., Gunetti, D., Picardi, C.: User authentication through keystroke dynamics. ACM Trans. Inf. Syst. Secur. (TISSEC) **5**(4), 367–397 (2002)
4. Chollet, F.: Keras (2015). https://github.com/fchollet/keras
5. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
6. Forsen, G.E., Nelson, M.R., Staron Jr., R.J.: Personal attributes authentication techniques. Technical report, DTIC Document (1977)
7. Gaines, R.S., Lisowski, W., Press, S.J., Shapiro, N.: Authentication by keystroke timing: some preliminary results. Technical report, DTIC Document (1980)
8. Giot, R., Hemery, B., Rosenberger, C.: Low cost and usable multimodal biometric system based on keystroke dynamics and 2D face recognition. In: ICPR (2010)
9. Gunetti, D., Picardi, C.: Keystroke analysis of free text. ACM Trans. Inf. Syst. Secur. (TISSEC) **8**(3), 312–347 (2005)
10. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: International Joint Conference on Neural Networks, IJCNN, vol. 1, pp. 593–605 (1989)
11. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
12. Killourhy, K.S., Maxion, R.A.: Comparing anomaly-detection algorithms for keystroke dynamics. In: DSN (2009)
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization (2014). CoRR abs/1412.6980
14. Le, Q.V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Ng, A.Y.: On optimization methods for deep learning. In: ICML (2011)
15. Leggett, J., Williams, G.: Verifying identity via keystroke characterstics. Int. J. Man Mach. Stud. **28**(1), 67–76 (1988)
16. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models (2013)

17. Monrose, F., Rubin, A.D.: Keystroke dynamics as a biometric for authentication. Future Gener. Comput. Syst. **16**(4), 351–359 (2000)
18. Peacock, A., Ke, X., Wilkerson, M.: Typing patterns: a key to user identification. IEEE Secur. Priv. **2**(5), 40–47 (2004)
19. Schapire, R.E.: A brief introduction to boosting. In: IJCAI (1999)
20. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
21. Syed, Z., Banerjee, S., Cheng, Q., Cukic, B.: Effects of user habituation in keystroke dynamics on password security policy. In: HASE (2011)
22. Syed, Z., Banerjee, S., Cukic, B.: Leveraging variations in event sequences in keystroke-dynamics authentication systems. In: HASE (2014)
23. Zhong, Y., Deng, Y., Jain, A.K.: Keystroke dynamics for user authentication. In: CVPR (2012)