# Biometric Keystroke Learning for User Authentication

**Ryan Kearns** (kearns@stanford.edu)
**Alex Langshur** (adl@stanford.edu)
**Harry Mellsop** (hmellsop@stanford.edu)
github.com/alangshur/biometric-typing
CS221: AI Principles and Techniques
Stanford University

## Abstract

Through research, we ascertained that typing habits vary subtly between individuals. These habits can be considered to be a unique 'biometric signature' of a given person, and therefore enable us to discern between people given a sample of their typing.

In this report, we describe the approach that we used to create an augmented password security system, based heavily on this idea. The key goal is to not only verify whether a given password has been correctly entered, but also to classify whether or not the typist is the real user.

We experimented with various models in an attempt to discover which performed best at determining the authenticity of the user. Our initial attempts relied on Manhattan and Euclidean distance inferencing techniques, from which we observed promising results, particularly from Manhattan.

To do better, we applied an Adam-optimized logistic regression, which created a high-dimensional hyper-ellipsoidal decision boundary, classifying whether or not a given user was authentic. Our results indicated that was could classify a user with an accuracy of 92%, with a false positive rate of 1.2%, and precision of 82%. These statistics are extremely promising, and improve upon those reported in Killourhy et. al.'s 2009 paper. As such, we believe that at scale this system could provide a meaningfully augmented security system, with a degree of redundancy created even when a user's password has been compromised.

## 1 Introduction

While passwords form the backbone of most modern system security implementations, they are far from perfect. Every second, 92 passwords are stolen [2], and typically this theft represents a total compromise of the associated account, with over 2.2 billion online accounts compromised due to password leaks in 2017 [5].

Password systems that rely purely on the password being accurately typed are imperfect, and as such, we wanted to investigate ways to improve the security of password systems using the "biometric signature" of a given user. Through research, we discovered that the way in which people type is extremely unique, and could therefore theoretically be of huge value in identifying the authenticity of a given typist in a password context.

Ideally, we want to create a system where when a given typist enters a known password, we can classify them as authentic or inauthentic simply based on the manner in which they typed.

## 2 Previous Work

Forsen et al. were the first to investigate whether users could be differentiated by typing sequences [4]. Robert Moskovitch et. al. further formalized the use of behavioral biometrics for password security in a 2009 paper presented to the IEEE International Conference on Intelligence and Security Informatics. The publication was an early effort to introduce biometric security measures that relied on present hardware rather than new systems. Moskovitch and colleagues identified both keyboard and mouse biometrics, including factors like flight time, dwell time, frequency of typing errors, and use of particular control keys [11].

Early efforts in this paradigm had already seen reasonable success. Sungzoon Cho and colleagues were able to achieve a 0.0% false acceptance rate (FAR) and a 1% false recognition rate (FRR) using a multilayer perceptron neural network to differentiate a user from an imposter typist [3]. Daw-Tung Lin saw similar results (FAR = 0.0%, FRR = 1.1%) using a similar neural network approach [9].

Biometric keystroke identification has since been employed in a number of applications in the last few years. Coursera, one of the largest massive open online course (MOOC) vendors with over 35 million users, used keystroke ID technology to guarantee valid certificates for students, which students could then show to prospective employers [1]. They describe their “Signature Track” system in a 2013 paper, in which students type a short phrase to authenticate their identity before submitting an online assignment [10]. Signature Track was discontinued in 2017, allegedly due to inaccuracies in prediction and reports from students that the procedure was inconvenient.

However, these implementations and papers all attempted to classify individuals based on general typing. With specific respect to biometric security implementaions in password security, we look to the 2009 paper “Comparing Anomaly-Detection Algorithms for Keystroke Dynamics,” from Kevin S. Killourhy and Roy A. Maxion, where they evaluated 14 implementations of keystroke biometric algorithms on biometric password security. We aligned our paper’s motivation with Killourhy and Maxion’s in this paper. Their explicit objective was to distinguish between password attempts from a genuine user and an imposter using biometric data. Additionally, Killourhy and Maxion mentioned that the noisiness of different data collection schemas and experimental techniques made comparisons of different models difficult at their time of writing. This being the case, they intended their study as a “benchmark” with which to gauge future progress on anomaly-detection algorithms [6]. The work also provided the CMU Keystroke Dynamics Benchmark Dataset that we use in our experiment. Killourhy and Maxion found the scaled Manhattan distance and the Mahalanobis distance nearest neighbours inferencing techniques to be their top two performing detectors.

## 3 Data

Raw keystroke data is parameterized by a sequence of key events and their durations. For a sequence of $n$ keystrokes, we collect three data points for each pair of adjacent keys $k$ and $k+1$ in a sequence. These are: the ‘Hold’ time, from when key $k$ is pressed to when key $k$ is released; the ‘Up-Down’ time, from when key $k$ is released to when key $k+1$ is pressed; and the ‘Down-Down’ time, from when key $k$ is pressed to when key $k+1$ is pressed. We adopted this scheme from Killourhy and Maxion's paper to correspond with their dataset.

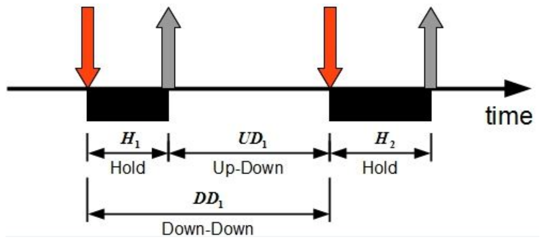

Figure 1: Keystroke data for a two-character sequence.

In the case of Figure 1 above, $H_1$, $DD_1, UD_1$, and $H_2$ all represent times in seconds, and are stored along with the names of the involved keyboard keys in our data set. For a trivial case where our password is `matt` (the name of our computationally challenged roommate, who participated in no further way to this project), the data is represented in code as the following. Here, $t_{ij}$ denotes the value of data point $j$ of character index $i$ for the password example $s =$ `"matt"`:

$$\text{Data}(s) = \Big\{(m, \text{None}, H) : t_{11}, (a, m, DD) : t_{12}, (a, m, UD) : t_{13}, \ldots, (t, t, UD) : t_{43}\Big\}$$

If there are no repeated sequences of 2 characters in the string, we find that the size of the data for one attempt will be $\big|\text{Data}(s)\big| = 3 \cdot \texttt{len}(s)$. For passwords that repeat features, such as `"mattt"`, the duplicate features will map to the mean of all matching keystoke pattern times. In these cases, $|\text{Data}(s)| < 3 \cdot \texttt{len}(s)$, though for reasonably non-repetitive passwords we do not anticipate this to be an issue. In our case, the password we used (`.tie5Roanl`) for training and testing had no repeated sequences.

We intend to collect a series of password attempts like these for each user in our experiment, so that each user $u$'s biometric data will be fully encapsulated in a list $(\text{Data}_{u,1}(s), \ldots, \text{Data}_{u,n}(s))$. Note that this data set is not yet the feature set for an attempt; we will have to normalize and add some additional non-linear features, which we describe in **3.3**.

## 3.1 Datasets

We were fortunate to have access to a comprehensive online dataset to provide a proof-of-concept for our initial modelling and learning. The CMU Keystroke Dynamics Benchmark Dataset contains password attempts from 51 users, each of whom typed the password `.tie5Roanl` 400 times. We downloaded the dataset and parsed it to populate our original feature vectors. This data is specific to a singular password and key sequence (e.g., the participants had to type the exactly correct keystrokes or the sample would be rejected). We wanted our model to work for any potential user's password, and also to tolerate erroneous attempts at the password and imperfect keystroke sequences. We collected our own data to have additional passwords with which to train the model, which will be discussed in **3.2**.

## 3.2 Manual Data Collection

Further to the CMU dataset that we utilised, one of our key goals with this project was to create a system that a user could take, train with their own typing habits, and then have it accept only them moving forward.

In order to achieve this, we had to gather biometric typing data from an arbitrary password entry attempt from a user. This data can then be parsed into a feature vector, as explained below, and then either used for training or testing purposes.

In order to do this, we built a simple keylogging piece of software using Python's `pynput` library. This task is surprsingly non-trivial, as we had to account for creating clean data, in the same format as our CMU data, where users might elect to use the shift key, caps-lock key, push a new key before releasing the previous key, and so on. Eventually, however, we arrived at a working version of this keylogger, which can be found in `data/userInterface.py` in our respository.

### 3.3 Feature Extraction

Given an attempt of the password $s$ by some user, parsed according to Data($s$), our next step is to extract features using $\phi(\text{Data}(s))$. This attempt could come from the dataset or our manual data collection interface, and should be handled in identical fashion. Our model is to be trained with respect to a single genuine user. Once trained, we can test the model against imposter's key sequences.

The first important consideration is to think about the decision boundary that our feature templates are creating. With standard linear features, whatever classifier we decided to use would effectively be training a hyperplane decision boundary parameterized each key event in the password sequence. This approach is suboptimal, as each feature should not linearly contribute to determining whether or not a user is genuine. For example, a 2x increase in typing speed from user $a$ to user $b$ should not increase or decrease the component likelihood that $a = b$, or $a \neq b$.

In order to minimise training load, we elected to normalize each data entry with respect to the average of the real user's password attempts, such that the set of all feature vectors would be centered about the origin for a given user. Recall that the data set Data($s$) is comprised of mappings from all $(k, k+1, e \in \{H, DD, UD\})$ to corresponding $t_{ij}$'s. For each $t_{ij}$ in one instance of Data($s$), let $\hat{t_{ij}}$ represent the average of all such times across all attempts. The feature extractor $\phi$ adds a linear and squared feature for each key event, which now map to the deviation of their attempt's time from the average. We also add a 1-feature for stability. All in all, a feature vector for one instance of Data($s$) looks like this (again, let's assume $s = \texttt{matt}$ for simplicity):

$$\phi(\text{Data}(s)) = \Big\{ (\texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}) : 1, (m, \texttt{None}, H, \texttt{`linear'}) : t_{11} - \hat{t_{11}},$$

$$(m, \texttt{None}, H, \texttt{`squared'}) : (t_{11} - \hat{t_{11}})^2, \ldots, (t, t, UD, \texttt{`squared'}) : (t_{43} - \hat{t_{43}})^2 \Big\}$$

The presence of squared features allows for the model to learn a variable-radii hyper-ellipsoidal decision boundary for one user's sequence of password attempts, providing finer tuning than the aforementioned hyperplane decision boundary. If some attempt has dimensions that deviate significantly from the learned averages $\hat{t}$, our model should flag it as an imposter. Essentially, if the feature vector of a particular user is within a trained high-dimensional hyper-ellipsoid, we can accept it as genuine. Because we normalise these vectors, we can safely accept everything within this boundary.

## 4 Models

Our plan from project start was to build a trivial inference tool and then manually implement a much more powerful mechanism that blends results from the simplified models and more advanced techniques. This approach allows us to gather further insight into the nature of the problem we are tackling, and ideally allow us to arrive at a better solution more quickly. Thus, our discussion of the models used in this report will be tackled in two parts; first, an explanation of the initial, naive modelling attempts, and, second, an in-depth derivation of a more advanced modelling attempt.

### 4.1 Naive Modelling

In our initial modelling attempts, we trialed several different naive models in order to effectively discern the difference between the valid user and any invalid attempts at entering the valid user's password. As a baseline algorithm to carry out this task, we started by implementing a simplified binary version of nearest neighbors using Manhattan and Euclidean distances.

For both of these models, we are provided a collection of training data that consists solely of password attempts belonging to the valid user. Let us refer to the raw valid data as $n$ inputs $\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}$, which spawn the $n$ formatted data vectors $\phi(\vec{x}^{(1)}), \ldots, \phi(\vec{x}^{(n)})$ when processed with the feature extractor $\phi$ defined in the **Data** section.

For both the Manhattan and Euclidean models, we can start by writing the valid mean data vector when trained on $\phi(\vec{x}^{(1)}), \ldots, \phi(\vec{x}^{(n)})$ as the following:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \phi(\vec{x}^{(n)})$$

Next, we can look into the process of inference over these two trained models. This process is executed in different ways for the Manhattan and Euclidean models.

### 4.1.1 Manhattan Inference

In the Manhattan model, we will take in a password data vector $\vec{x}^{(u)}$ from an arbitrary user (valid or invalid), process this data through the feature extractor $\phi$ which provides the processed feature vector $\phi(\vec{x}^{(u)})$, and calculate the Manhattan distance from $\phi(\vec{x}^{(u)})$ to the trained valid mean $\hat{\mu}$. This can be done as follows (note that $m = |\hat{\mu}|$):

$$f_{\text{Manhattan}}(\phi(\vec{x}^{(u)})) = \Big|\,\hat{\mu}_1 - \phi_1(\vec{x}^{(u)})\,\Big| + \cdots + \Big|\,\hat{\mu}_m - \phi_m(\vec{x}^{(u)})\,\Big|$$

Given a hyperparameterized threshold $T$, we will classify whether the data $\phi(\vec{x}^{(u)})$ is associated with the valid user as follows ($\hat{y} = 1$ if the user is valid and $\hat{y} = 0$ otherwise):

$$\hat{y}(\phi(\vec{x}^{(u)})) = \mathbb{1}\left[f_{\text{Manhattan}}(\phi(\vec{x}^{(u)})) < T\right]$$

Our approach in in determining the hyperparameter $T$ is outlined in **6.1**.

### 4.1.2 Euclidean Inference

Note that inference over the Euclidean model is closely related to inference over the Manhattan model. For a feature extracted input vector $\phi(\vec{x}^{(u)})$, we can calculate the squared-Euclidean distance from this input to the trained valid mean $\hat{\mu}$ as follows (note that $m = |\hat{\mu}|$):

$$f_{\text{Euclidean}}(\vec{x}) = (\hat{\mu}_1 - \phi_1(\vec{x}^{(u)}))^2 + \cdots + (\hat{\mu}_m - \phi_m(\vec{x}^{(u)}))^2 = \big\|\vec{\mu} - \phi(\vec{x})\big\|^2$$

Given a hyperparameterized threshold $T$, we will again classify whether the data $\phi(\vec{x}^{(u)})$ is associated with the valid user as follows:

$$\hat{y}(\phi(\vec{x}^{(u)})) = \mathbb{1}\left[f_{\text{Euclidean}}(\phi(\vec{x}^{(u)})) < T\right]$$

Our approach in in determining the hyperparameter $T$ is outlined in **6.1**.

## 4.2 Advanced Modelling

While the two simplified modelling attempts were an effective initial approach, our plan from project start was to build a trivial inference tool and then manually implement a much more powerful mechanism that blends insight from the simplified models and more advanced modelling decisions. To carry this out, we will move away from the binary clustering classification architecture adopted in the naive models and instead implement an architecture based on support-vector machining.

After observing the way input into the naive models clustered around the mean feature-extracted data vector for the valid user, we agreed that a hyper-ellipsoidal decision boundary could act as the basis for a more powerful inference mechanism. As explained in the feature extraction section, this non-linearity can be accomplished using a feature extractor $\phi$ over a supervised input data point $(\vec{x}^{(i)}, y^{(i)})$ to produce the following data vector:

$$\phi(\vec{x}^{(i)}) = [1, x_1, \ldots, x_{|\vec{x}|}, x_1^2 + \cdots + x_{|\vec{x}|}^2]$$

Next, we can define the decision boundary of the inference mechanism based on the following hyperplane parameters – the trained weights vector $\vec{w}$, the predictor function $f$, and the threshold value $T$:

$$\{\, \vec{x} \in \mathbb{R}^{|\vec{w}|} \mid f\big(\vec{w}^{\,T}\phi(\vec{x}^{(i)})\big) = T \,\}$$

This hyperplane equation suggests that positive decisions should be made for $f\big(\vec{w}^{\,T}\phi(\vec{x}^{(i)})\big) > T$ and negative decisions should be made for $f\big(\vec{w}^{\,T}\phi(\vec{x}^{(i)})\big) < T$. While there are many effective candidates for the predictor function $f$, we will use logistic regression due to its relevant probabilistic interpretation among other critical characteristics. This particular model is based on the following logistic function:

$$\sigma(z) = (1 + \exp(-z))^{-1}$$
$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Using this particular predictor function $f$ will have several important benefits. First, it will produce a clear mapping of our predicted scores to the range $(0, 1)$, which allows us to easily trial different threshold values in that range. Moreover, it will incentivize a weight vector that produces scores further from the threshold value – this is particularly important in user authentication since we want the inference mechanism to make distinct decisions. We can therefore write our predictor function with input $\vec{x}^{(i)}$ as the following (note that, as defined by the feature list above, we already include a constant for bias and this doesn't need to be included explicitly in the logistic predictor):

$$f_{\vec{w},T}(\vec{x}^{(i)}) = (1 + \exp(-\vec{w}^T\phi(\vec{x}^{(i)})))^{-1}$$

Given supervised training data $(\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(n)}, y^{(n)})$, we require a method that trains the weight vector $\vec{w}$ such that new input data $\vec{x}^{(i)}$ will produce $\hat{y}^{(i)} = 1$ if the input data belongs to the valid user and $\hat{y}^{(i)} = 0$ if the input data belongs to an invalid user. Using the underlying probabilistic interpretation of logistic regression, we can write the probability of occurrence for a single data point as the following:

$$P(Y = \hat{y}^{(i)} \mid X = \vec{x}^{(i)}) = \sigma(\vec{w}^T\phi(\vec{x}^{(i)}))^{\hat{y}^{(i)}} \cdot \left[1 - \sigma(\vec{w}^T\phi(\vec{x}^{(i)}))\right]^{(1-\hat{y}^{(i)})}$$

Using this expression for a single data point, we can derive a log-likelihood expression for all of our training data $(\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(n)}, y^{(n)})$:

$$\mathcal{L}(\vec{w}) = \prod_{i=1}^{n} P(Y = \hat{y}^{(i)} \mid X = \vec{x}^{(i)}) = \prod_{i=1}^{n} \left[\sigma(\vec{w}^T\phi(\vec{x}^{(i)}))^{\hat{y}^{(i)}} \cdot \left[1 - \sigma(\vec{w}^T\phi(\vec{x}^{(i)}))\right]^{(1-\hat{y}^{(i)})}\right]$$

$$\ln \mathcal{L}(\vec{w}) = \sum_{i=1}^{n} \left[\hat{y}^{(i)} \ln \sigma(\vec{w}^T\phi(\vec{x}^{(i)})) + (1 - \hat{y}^{(i)}) \ln\left(1 - \sigma(\vec{w}^T\phi(\vec{x}^{(i)}))\right)\right]$$

Since the logarithm is a monotonically increasing function, maximizing the general probabilistic likelihood function over the training data is equivalent to maximizing the log-likelihood function. In order to solve for the globally optimal $\vec{w}$, we can therefore introduce the following optimization problem:

$$\vec{w}^* = \arg\max_{\vec{w}} \ \ln \mathcal{L}(\vec{w})$$

$$\vec{w}^* = \arg\max_{\vec{w}} \ \sum_{i=1}^{n} \left[\hat{y}^{(i)} \ln \sigma(\vec{w}^T\phi(\vec{x}^{(i)})) + (1 - \hat{y}^{(i)}) \ln\left(1 - \sigma(\vec{w}^T\phi(\vec{x}^{(i)}))\right)\right]$$

### 4.3 Adam-Optimized Logistic Stochastic Gradient Ascent

To efficiently solve this non-convex optimization problem over a large training set, we turned to stochastic gradient ascent (note that we're working with an ascent algorithm rather than descent due to the maximization problem). Additionally, due to the complex nature of the problem, we built a full implementation of Adam optimization, which is considered to be one of the best (if not the single best) SGA optimizers available [7].

In Adam optimization, we adapt the learning rate $\eta$ of SGA based on exponentially-decaying first and second gradient moments. By tracking change in the gradient mean, as well as its uncentered variance, Adam is able to optimize learning rate based on the momentum (how well it is performing at a given iteration) of the algorithm.

For this version of stochastic gradient ascent, we keep track of a default step size $\eta$, exponential decay rates $\beta_1, \beta_2$ for the first and second moments, respectively, the current SGA epoch $t$, and moment vectors $\vec{m}, \vec{v}$ for the first and second moments, respectively (which we initialize to zero).

At an arbitrary epoch $t = k$, we start by updating $t$ to $t = k + 1$ and then derive the gradient of our stochastic objective with respect to $\vec{w}$ for a single training data point $j \in \{0, \dots, n\}$:

$$\vec{g}_{t=k+1} = \nabla_{\vec{w}} \ln \mathcal{L}_j(\vec{w}_{t=k}) \underset{i \in \{1,\dots,|\vec{w}|\}}{\longrightarrow} \frac{\partial \ln \mathcal{L}_j(\vec{w}_{t=k})}{\partial \vec{w}_i} = \left[y^{(j)} - \hat{y}^{(j)}\right] x_i^{(j)}$$

$$= \left[y^{(j)} - \sigma(\vec{w}_{t=k}^T \phi(\vec{x}^{(j)}))\right] x_i^{(j)}$$

Next, we update our biased first and second moment estimates using $g_{t=k+1}$:

$$m_{t=k+1} \longrightarrow \beta_1 \cdot m_{t=k} + (1 - \beta_1) \cdot g_{t=k+1}$$

$$v_{t=k+1} \longrightarrow \beta_2 \cdot v_{t=k} + (1 - \beta_2) \cdot g_{t=k+1}$$

With the biased estimates, we can compute the bias-corrected first and second moment estimates:

$$\hat{m}_{t=k+1} = \frac{m_{t=k+1}}{1 - \beta_1^t}, \quad \hat{v}_{t=k+1} = \frac{v_{t=k+1}}{1 - \beta_2^t}$$

Finally, we update our $\vec{w}$ estimate for the current epoch (where $\epsilon$ is a small constant to prevent division-by-zero errors):

$$\vec{w}_{t=k+1} = \vec{w}_{t=k} + \eta \cdot \frac{\hat{m}_{t=k+1}}{\sqrt{\hat{v}_{t=k+1}} + \epsilon}$$

After tuning the Adam-optimization hyperparameters, we found that their optimal values are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.01$, and $\epsilon = 10^{-8}$.

## 5 Results

To test the accuracy of our model, we used our CMU dataset entirely. We ran our model, attempting to classify 400 'real-user' password attempts and 20,000 adversary attempts.

Assessing accuracy on user-generated data was more difficult, as it was challenging to gather statistically meaningful data on real people, but we will discuss results pertaining to this in the **Evaluation** and **Future Work** sections.

### 5.1 Accuracy

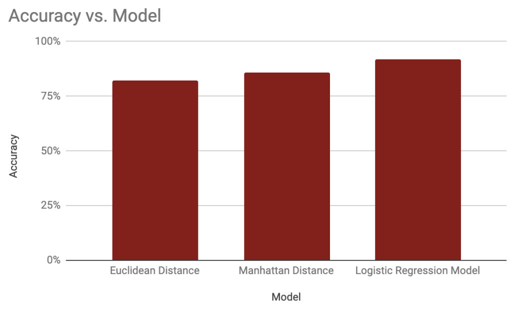

Figure 2: Accuracy ratings for three models.

Accuracy is simply defined as the percentage of examples that the classifier in question correctly classified.

Our Euclidean Nearest-Neighbours implementation was 82% accurate. Manhattan distance Nearest-Neighbours performed slightly better, with 86%. However, our Adam-optimized Logistic Regression approach was notably better, with 92% classification accuracy.

## 5.2 Precision

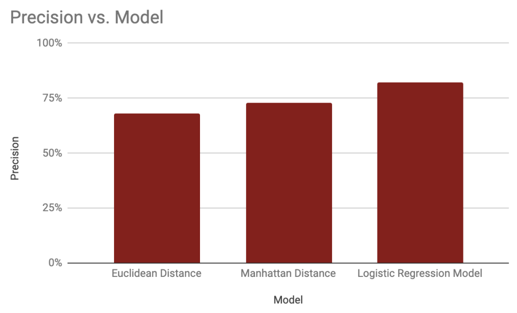

Figure 3: Precision ratings for three models.

Precision is defined as the percentage of password attempts classified as positive, that really were positive. As such, precision is an important metric to assess as it needs to be carefully balanced against recall for ideal real-world classification performance. [8].

With respect to precision, Euclidean again performs the worst, with 68% precision. Manhattan is slightly better, with 73%. As with accuracy, our Adam-optimized logistic regression model performs the best, with a precision of 82%.

## 5.3 Recall

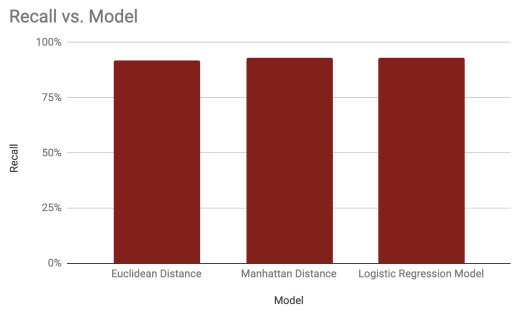

Figure 4: Recall ratings for three models.

Recall is defined as the percentage of password attempts that are positive, that were classified as positive by the classifier. This statistic is extremely important, as it determines how many of our valid user entries we accept into the system.

Here, all of our models perform well. Euclidean returns a recall of 92%, while both Manhattan Distance and our Adam-optimized Logistic Regression return a recall of 93%.

## 5.4 False Positive Rate

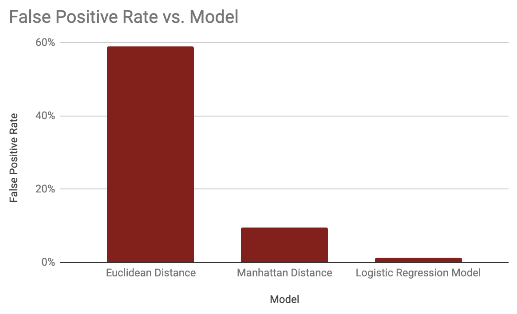

Figure 5: False positive ratings for three models.

False positive rates are where our models differ most dramatically. As is the theme, Euclidean Distance returned the worst false positive rate, with a 59% false positive rate. Clearly, this is unacceptably bad for a model, as this means that 59% of negative events were actually, falsely, categorised as positive.

In contrast, Manhattan Distance returned a far better false positive rate, with only 9.6%. However, our Adam-optimized Logistic Regression model was even better, with a 1.2% false positive rate.

## 6 Evaluation and Error Analysis

### 6.1 Naive Error Evaluation

One of the critical parameters for evaluating model performance, due to the stringent requirement that no invalid user receive a valid classification, is the false-positive rate (ratio between number of incorrect positive predictions and total number of true negative outcomes in the test set). In our initial modelling attempts, we found that the naive clustering architecture of the Euclidean and Manhattan models provided little support for the maintenance of low false-positive rates. Upon observing the high false-positive rates displayed in section **5.4** above, we attempted to manually lower the threshold distance $T$ required for a positive classification in the naive models:

$$\hat{y}(\phi(\vec{x}^{(u)})) = \mathbb{1}\big[f(\phi(\vec{x}^{(u)})) < T\big]$$

Perfectly aware that this may also lower overall accuracy, we found that, when manually tuning the value of $T$, the accuracy-to-false-positive trade-off was significantly skewed against high accuracy ratings. In other words, a small decrease in the acceptable threshold distance $T$ resulted in a disproportionately large drop in the overall test accuracy.

Regardless of the poor false positive ratings of the naive models, we nonetheless found that the Manhattan distance model produced a much lower false positive rate than the Euclidean distance for optimal values of $T$. To make sense of this, first note that the feature-extracted data is roughly on the order of magnitude of $10^{-1}$. Hence, for a single component in the calculation of the Euclidean distance between a feature-extracted vector and the valid mean vector, the difference will be on this same order of magnitude. If we square this difference, as we do in the Euclidean model, the difference will decrease further since it is strictly less than one. In doing this, we are effectively "cheating" a smaller distance between data vectors, which is more likely to result in a positive classification and increase the false positive rate. Since we don't carry out this squaring operation in the Manhattan model, we avoid this "cheating" phenomenon and find a much lower false-positive rate. For this same reason, we find that the accuracy of the Manhattan model is slightly higher than the Euclidean model; clearly, a lower false-positive rating influences a higher accuracy score.

Since, under the geometric interpretation of the naive models, both clustering architectures are based on a form of linear distance from the input data vector to the valid mean vector, we hypothesized and were correct in finding that the accuracy, precision, and recall ratings between the two naive models are close in value.

### 6.2 Logistic Error Evaluation

In order to evaluate error in the more advanced logistic regression model, we provided a more versatile framework for modifying model and learning hyperparameters to produce better-valued results. Namely, we can use the probabilistic interpretation of logistic regression to determine the general likelihood of our training and testing data given a learned weight vector $\vec{w}$. This can be done by recycling our original likelihood function from **4.2** as follows:

$$\mathcal{L}(\vec{w}) = \prod_{i=1}^{n} P(Y = \hat{y}^{(i)} \mid X = \vec{x}^{(i)}) = \prod_{i=1}^{n} \Big[\sigma(\vec{w}^T\phi(\vec{x}^{(i)}))^{\hat{y}^{(i)}} \cdot \big[1 - \sigma(\vec{w}^T\phi(\vec{x}^{(i)}))\big]^{(1-\hat{y}^{(i)})}\Big]$$

By iterating through our entire training and test set with a learned value of $\vec{w}$, we are able to derive the general likelihood of that data. By tweaking threshold values in the model, coefficient parameters in the logistic function, and hyperparameters from Adam-optimized stochastic gradient descent, we are able re-learn a weight vector $\vec{w}$ and quickly evaluate its performance using the general train/test data likelihood.

Using this error analysis technique, we were able to achieve higher accuracy than the naive models while ensuring a far lower false positive rate of approximately $1\%$. If we compare the geometric interpretation of the naive models and that of logistic regression, we find that the clustering architecture of the Euclidean and Manhattan distances can be approximately interpreted as hyperspherical decision boundaries. However, this proves to be wildly limiting due to the fixed higher-dimension "radius" of this boundary – the radius threshold $T$ is constant for every distance calculation between an input vector and the valid mean vector. Conversely, in the feature extracted data vector $\phi(\vec{x}^{(i)})$ for logistic regression, we isolate squared features for each component in $\vec{x}^{(i)}$, which, geometrically, allows us to learn a hyperellipsoidal decision boundary with a variable "radius" length in each dimension. This not only allows us to fit a more personalized decision boundary to the valid user's data, but also ensures that less non-valid input data is classified as valid, thus resulting in lower false positive rates.

## 7 Future Work

There is additional work to be done across all facets of this project, and the field of behavioral biometric typing in general. We will discuss these in the same order as they came up in the paper.

### 7.1 Cleaner and Expanded Data Collection

As mentioned earlier in our analysis, our user-collected data was difficult to compare to the CMU dataset, due to the noisy conditions in which the data was collected. We ended up using data entries solely from the dataset for our final model results. In future attempts, we will want to tighten our own data collection scheme. Doing so should allow us to test with more flexibility than the CMU dataset allowed. The CMU dataset was produced in controlled conditions using a fixed password and hardware. Further, the password `.tie5Roanl` is designed to span the whole QWERTY keyboard layout. In reality, users type their passwords on a variety of different types of keyboards across different devices, and do not often design their passwords to be difficult to enter. One concrete next step would be testing our model's effectiveness across different hardware and with both shorter and longer passwords.

Additionally, we trained and tested our model on a relatively small dataset, with only around 20,000 examples to work with. We could likely improve our performance by collecting more data in a greater variety of settings. This procedure would also give our participants more practice with the password in question, leading to more realistic attempts.

### 7.2 Neural Network Modelling

In the future, we could undertake more ambitious modelling attempts to further define the decision boundary learned by our model. A finer decision boundary would allow for further scalability, which would be critical for this model's applicability to real-world situations.

Specifically, we would hope to utilize a neural network structure for more optimal non-linear feature extraction. Currently, we use squared deviation from the mean as our set of non-linear features. A hidden-layer perceptron model could learn more sophisticated non-linear features that would undoubtedly improve performance. There is evidence for this in a number of papers that achieved better results using neural network structures [3][9].

Figure 6: User training our biometric model.

## References

[1] Coursera, 2019, https://blog.coursera.org/about/.

[2] J. Brown. An average of 95 passwords are stolen every day, 2017, https://www.ciodive.com/news/an-average-95-passwords-stolen-per-second-in-2016-report-says/435204/.

[3] S. Cho, C. Han, D. H. Han, and H.-I. Kim. Web based keystroke dynamics identity verification using neural network, 2000, https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=5BADA02DC9A60E46B907A59EAE39C80F?doi=10.1.1.95.4863rep=rep1type=pdf.

[4] G. Forsen, M. Nelson, and J. R. Staron. Personal attributes authentication techniques. technical report radc-tr-77-333, 1977.

[5] T. Hunt. 86% of passwords are terrible (and other statistics), 2018, https://www.troyhunt.com/86-of-passwords-are-terrible-and-other-statistics/.

[6] K. S. Killourhy and R. A. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics, 2009, https://www.cs.cmu.edu/ keystroke/KillourhyMaxion09.pdf.

[7] D. Kingma. Adam: A method for stochastic optimization, 2014, https://arxiv.org/abs/1412.6980.

[8] W. Koehrsen. Beyond accuracy precision and recall, 2018, https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c.

[9] D.-T. Lin. Computer-access authentication with neural network based keystroke identity verification, 1997.

[10] A. Maas, C. Heather, C. T. Do, R. Brandman, D. Koller, and A. Ng. Moocs and technology to advance learning and learning research: offering verified credentials in massive open online courses, 2014, 10.1145/2591684.

[11] R. Moskovitch, C. Feher, A. Messerman, N. Kirschnick, T. Mustafić, A. Camtepe, B. Löhlein, U. Heister, S. Möller, L. Rokach, and Y. Elovici. Identity theft, computers and behavioral biometrics, 2009, http://www.ise.bgu.ac.il/faculty/liorr/idth.pdf.