Introduction

Timing information of a keyboard user's keystrokes has been studied for about three decades. Keystroke dynamics (Keystroke Dynamics, 2019) refers to the automatic identification or confirmation of an individual based on the manner and the rhythm of them typing on a keyboard. Keystroke dynamics, also known as typing dynamics, is a behavioral biometric. Behavioral biometrics is the biometric factor describing 'something you do'. Generally, keystroke dynamics are less reliable than physiological biometrics.

In keystroke dynamics, a biometric template is used to identify an individual based on their typing pattern, the rhythm, and the speed of typing on a keyboard. The typical keystroke interval time, referred to as a digraph (J. M. Chang, 2013), is expressed as the time between typing two characters. A user's keystroke rhythms are distinct enough from person to person for use as biometrics to identify people. The raw measurements used for keystroke dynamics are dwell time and flight time. Dwell time is the time duration that a key is pressed. Flight time is the time duration in between releasing a key and pressing the next key.

No additional equipment is required for keystroke dynamics. A standard keyboard (A. A. Vyazigin, 2019) is enough, which makes the user recognition system relatively cheap. Because keystroke dynamics features templates are difficult to forge, keystroke dynamics are a growing research area (Lozhnikov, 2016).

Some companies like TypingDNA, ID Control and BehavioSec develop software products applying keystroke dynamics.

Database Description

The database of keystroke data used comes from K. Killourhy and R. Maxion (Killourhy, 2009) and consists of 51 subjects typing the same password ('.tie5Roanl'). The data in the database is the keystroke-timing information from these 51 subjects typing said password 400 times over 8 sessions.

The data includes the following keystroke information, where each following item represents one column of 34 from the table of the database: (1) a unique identifier for each typing subject, such as 's002', (2) a 'sessionIndex' which identifies the session in which the password was typed (ranges from 1 to 8 for each subject), (3) this column, 'rep', represents the repetition number of the password being typed within the session, (4) the following 31 columns of the table contain the timing information for the password being typed.

Table 1: Timing Database

Column	Representation
Col 1	Subject identifier
Col 2	Session (index) number
Col 3	Password repetition number
Col (H.*key*)	Hold time for *key*.
Col (DD.*k1*.*k2*)	Keydown of *k1*-to-keydown of *k1* time.
Col (UD.*k1*.*k2*)	Keyup of *k1*-to-keydown of *k2* time.

The last 31 columns of the database table, which house the timing information, are separated into three different types of information. The columns marked with an 'H.*key*' represent a hold time for the specified key (i.e. the time from when the subject pressed *key* to when it was released). The columns marked with 'DD.*key1*.*key2*' represent a time of keydown to keydown of *key1* and *key2* (i.e. the time from when the subject pressed *key1* to when they pressed *key2*). Lastly, the columns marked as ‘UD.*key1*.*key2*’ represent the keyup to keydown time of said keys (i.e. the time between the subject releasing *key1* to when they pressed *key2)*. The information of how this database organizes its data is better depicted in Table 1.

Methods

Our test system is comprised of several Python files:

Get_data.py: gets the keystroke data from the data directory. Parameters are keystroke data directory, a given session k, and another given session k2. Get_data.py then takes the k and k2 sessions for each subject, and records the data from those session files.

Enhance.py: can apply one of 4 different enhancement techniques to the template and query.

- *StandardScaler* scales by removing the mean and scales the data to unit variance. StandardScaler very sensitive to the presence of outliers.
- RobustScaler removes the median and scales the data according to the quantile range parameter. The IQR is the range between the 1st quartile and the 3rd quartile. RobustScaler is not very sensitive to the presence of outliers.
- *MinMaxScaler* rescales the data set such that all feature values are in the range [0, 1]. MinMaxScaler is very sensitive to the presence of outliers.
- *PCA* is the idea is that we can select and keep the most important data points, and throw away the ones that contribute mostly noise.

Matcher.py: takes in template data and labels, query data labels, the choice of classifier that will be used to match the data, and the amount of data that will be used from the template and query data. It then matches the data using the chosen classifier and outputs the genuine and imposter scores. Parameters are X1 which represents the template data gathered from each subject from a given session, y1 which represents the labels for the template data, X2 which represents the query data gathered from each subject in a different session than the data collected in X1, y2 which represents the labels for the query data, classifier is a given number that chooses 1 of 4 classifiers to compare the template and query data, and QUERY_SIZE and TEMPLATE_SIZE are the percentage of data that will be randomly chosen from the query data and template data respectively.

- *K-Nearest Neighbors* (Pedregosa, 2011) is a supervised learning classification method that is computed from finding the minimum distance from the query sample to the k-nearest training samples. Whichever class makes up the most of the nearest neighbors is selected as the output class.
- A *random forest* (Pedregosa, 2011) is a supervised learning classification method that operates by using several decision trees at training time. The classifier outputs the class that has to most ‘votes’ from each decision tree. Random forests improve upon decision trees, because decision trees have a habit of overfitting the training data.
- *Naive Bayes* (Pedregosa, 2011) is a supervised learning classification method that uses probability theory to predict classes. Naive Bayes is naive in the sense that it doesn’t consider that features could be dependent upon one another. Naive Bayes is based on applying Bayes’ theorem.
- *Decision Tree* (Pedregosa, 2011) is a classification method that uses a tree structure to model decision making. It works like a flow chart, where each decision leads down a

different path. The leaf nodes represent the class prediction.

Main.py: uses Get_data.py to gather keystroke data from 8 sessions from each of the 50 subjects, Enhance.py to choose 1 of 4 methods to enhance the data, Matcher.py to match the given amount of template data to the given amount of query data using the chosen classifier, and Performance.py to plot the genuine and imposter scores using score distributions, DET curves, and ROC curves. The EER's from the comparison of each session are then plotted against each other.

Results

In this experiment the algorithm was modified in several ways. The first change that was made was increasing the amount of templates that were randomly chosen for the queries to be compared to. This was based on the idea that if there are more templates to be compared to it is more likely that the query will find its match. The reason for using 75% of templates was to increase the likelihood of a match while not using too much overhead. This proved to be correct as it decreased the average EER from 88.89% to 71.02%

The next change that made was using a Naive Bayes classifier. This change was made because Naive Bayes is highly accurate when applied to big data and it is a linear classifier which makes it much faster than random forests and KNN (Glen, 2019). This also proved to be correct significantly reducing the average EER from 71.02% to 25.58%.

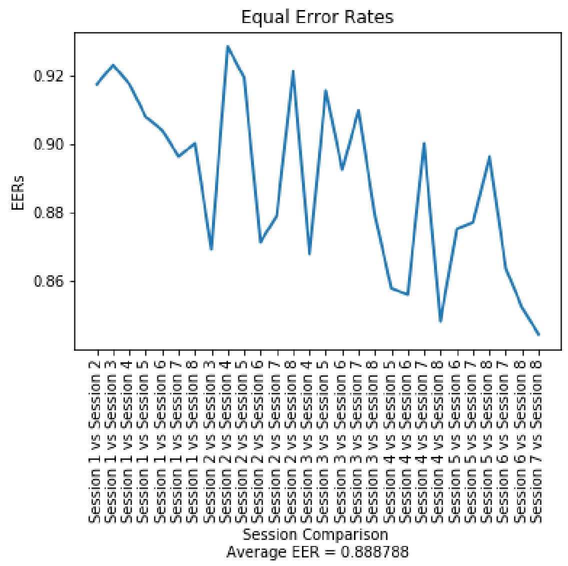
The next change that was made was using RobustScaler for the enhancer instead of StandardScaler. StandardScaler removes the mean and scales the data to unit variance which makes it very sensitive to outliers. This was not the best choice for the data as keystroke can be affected by many outside sources which could produce outliers in the data. RobustScaler removes the median and scales the data to the interquartile range (IQR), which is the range between the first and fourth quartile. Since the median and IQR are not sensitive to outliers this seemed to be the better option. This proved to be incorrect as the average EER slightly increased from 25.58% to 26.61%. This idea was revisited as we made other changes.

The next change we made was increasing the threshold value in Perfromance.py. The idea behind this change was that if a query must satisfy a greater criteria the false accept rate would decrease however this would come at the cost of an increased false reject rate. The threshold value was increased to 250 with very little effect slightly increasing the average EER from 25.58% to 25.88%. Based on these results, the next change was to decrease the threshold in order to decrease the false reject rate with the cost of an increased false accept rate. This produced slightly better results decreasing the average EER from 25.58% to 25.36%. Based on this result the threshold value was decreased again from 150 to 100. This increased the average EER from 25.36% to 26.24%. It was determined that the best results from this change came from decreasing the threshold value to 150 even though the results of all tests produced minimal change.

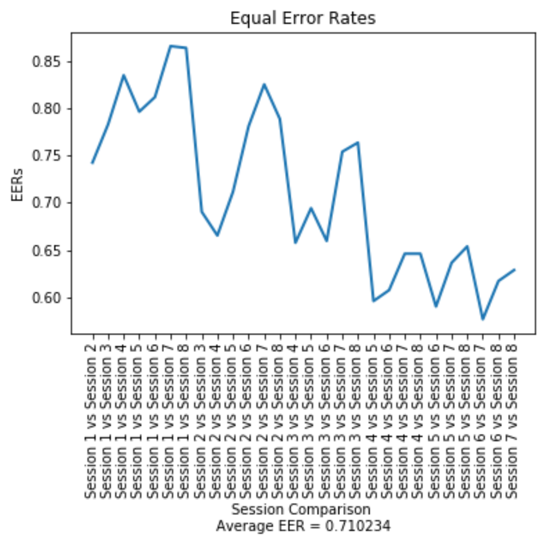
The final change that was made was filtering the features that were used for matching. Based on our keystroke data it can be seen that a wider range of data comes from keydown-keydown(DD) and keyup-keydown(UD) than hold time for each key, so the hold time was filtered out of the data. This change was not beneficial as it increased the average EER from 25.36% to 29.40%. After considering the range of data that was being used, the enhancer was changed from StandardScaler to RobustScaler. This produced better results than StandardScaler, however the average EER was still greater than the results with no feature filtering. This change proved to ineffective.

After the changes made to this algorithm, the best results were produced when using 75% of the template data, Naive Bayes classifier, StandardScaler, 150 threshold value, and no column filtering. The EER results from each change can be seen in the following charts:

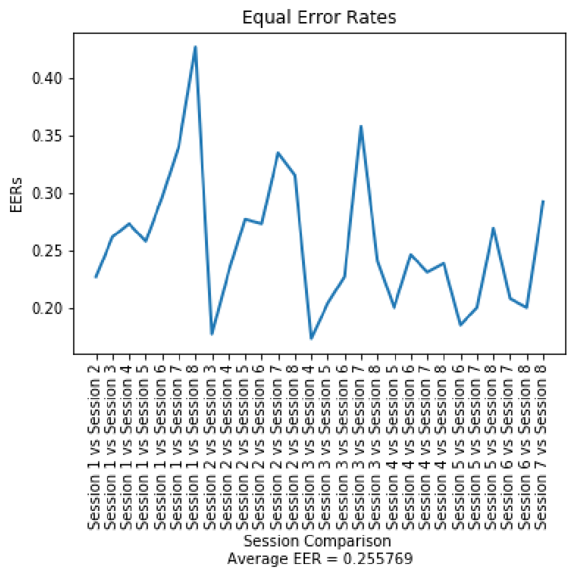
Default:



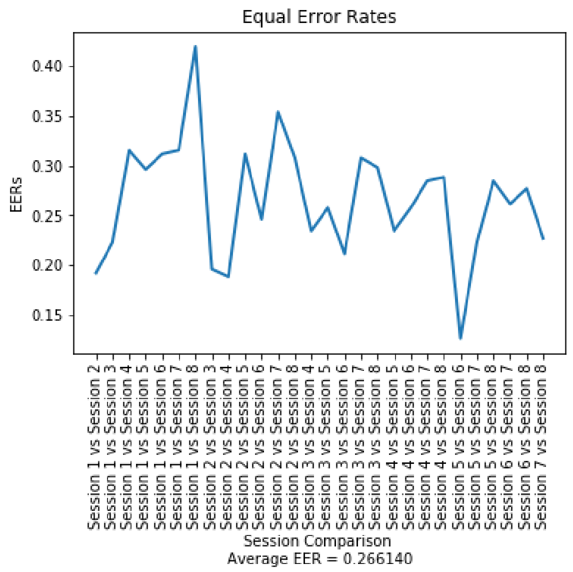
Change 1 - Template size



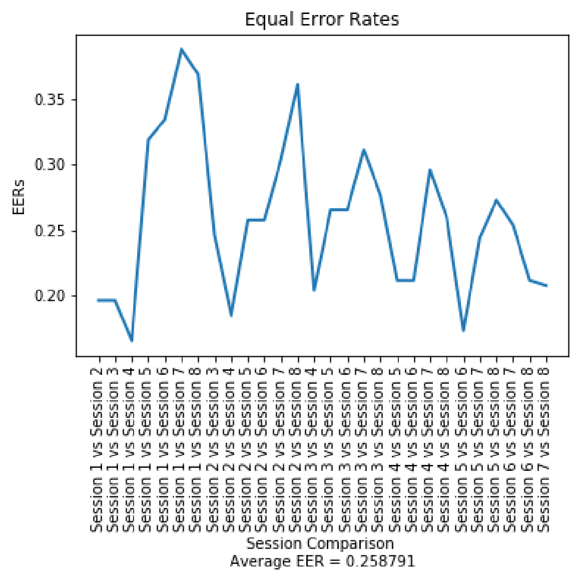
Change 2 - NB classifier



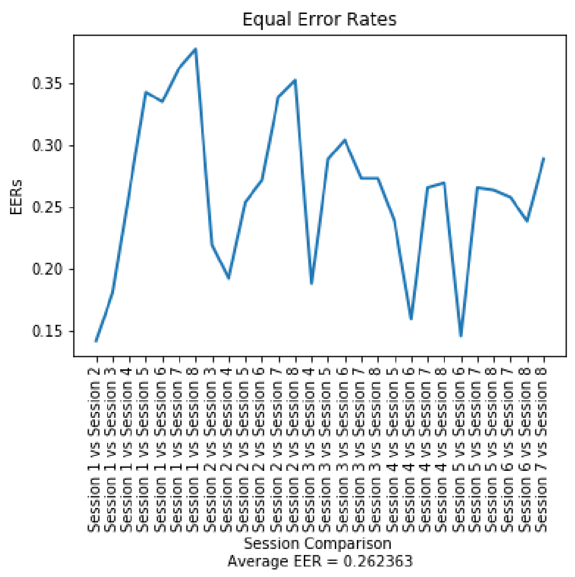
Change 3 - RobustScaler



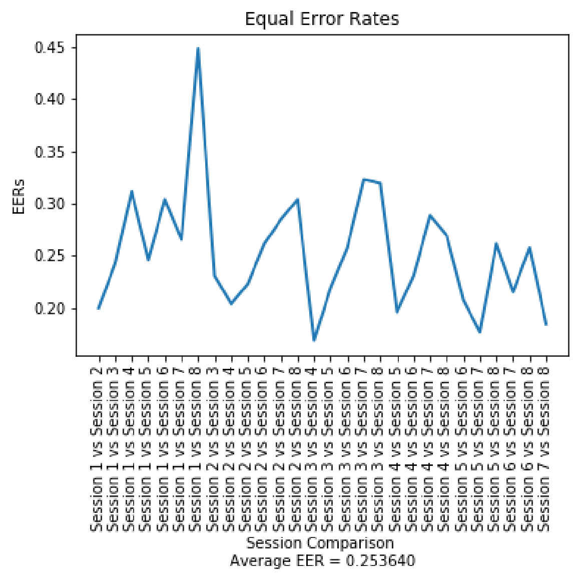
Change 4a - 250 Threshold



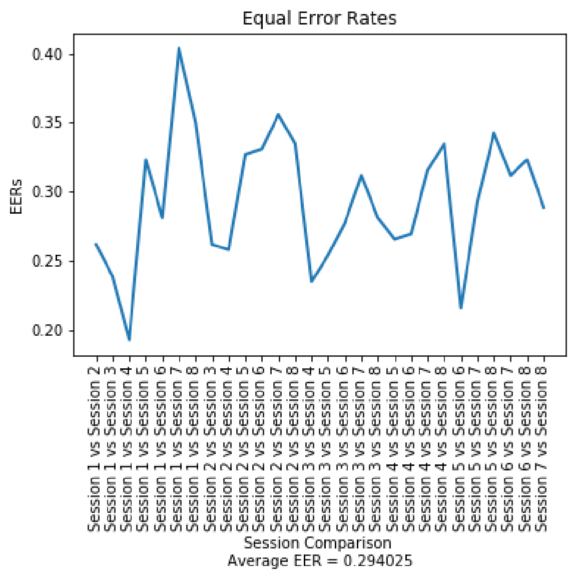
Change 4c - 100 Threshold:



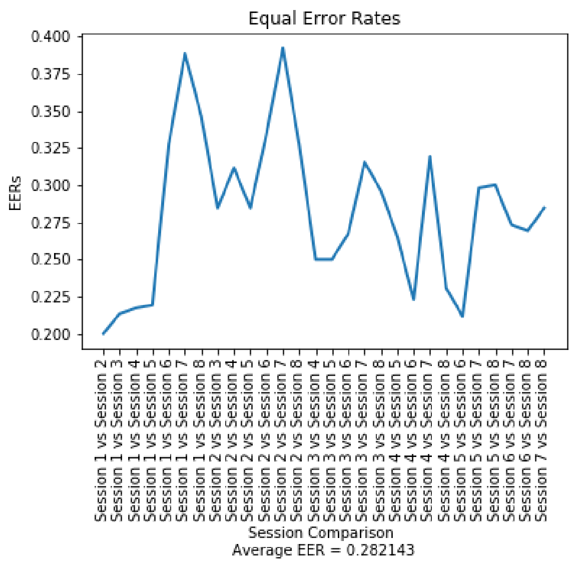
Change 4b - 150 Threshold



Change 5a - Column filtering with StandardScaler:



Change 5b - Column filtering with RobustScaler:



Summary

Some of the changes made in the keystroke authentication system positively affected the performance of said system. The first beneficial change came from modifying the amount of randomly chosen templates to be compared against the queries. Changing the template percentage to a value of 75% lowered our EER over 17%. Our second change, choosing to use a Naive-Bayes classifier, yielded a significant drop for the system's EER, dropping over 45% after the first change was made. We believe this is because of the advantage Naive-Bayes had over other classifiers, which is its accuracy and speed over a large amount of data.

The third change was of minimal effect on the performance of the system, which was changing the enhancing technique to the RobustScaler. We had thought changing the enhancing technique to one with less sensitivity of outliers. This had proved to be ineffective at performing better than the initial StandardScaler with Naive-Bayes.

The fourth change, changing the value of the threshold in the performance file, also proved to be ineffective in optimizing the system. The idea was that if a query satisfied more criteria the FAR would decrease, consequently, the FRR would increase because of this.

The fifth and final change, filtering the columns based on the features, also proved to be ineffective, even when tested with other enhancing techniques. The EER remained the same, or even worse at times, which showed filtering the columns in this manner wasn't necessary when trying to optimize the system.

For the future, there are other avenues we could take to improve the system. Overall we had reduced the average EER of the system over 60%. However, this does not qualify the system to be as optimized as it can be. The latter half of changes we made in attempt to optimize the system weren't as effective as hoped. However, there are many components to this system and other optimizations we may have missed could still improve the system.

References

A. A. Vyazigin, N. Y. Tupikina and E. V. Sypin, "Software Tool for Determining of the Keystroke Dynamics Parameters of Personal Computer User," *2019 20th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices (EDM)*, Erlagol (Altai Republic), Russia, 2019, pp. 166-171.

Admin. (n.d.). Keystroke Dynamics. Retrieved December 1, 2019, from http://www.biometric-solutions.com/keystroke-dynamics.html.

Glen, S. (2019, June 19). *Comparing Classifiers: Decision Trees, K-NN & Naive Bayes.* Retrieved from Data Science Central: https://www.datasciencecentral.com/profiles/blogs/comparing-classifiers-decision-trees-knn-naive-bayes

J. M. Chang *et al*., "Capturing Cognitive Fingerprints from Keystroke Dynamics," in *IT Professional*, vol. 15, no. 4, pp. 24-28, July-Aug. 2013.

Killourhy, K. S., & Maxion, R. A. (2009, June). Comparing anomaly-detection algorithms for keystroke dynamics. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks* (pp. 125-134). IEEE.

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

P. Lozhnikov, E. Buraya, A. Sulavko and A. Eremenko, "Methods of generating key sequences based on keystroke dynamics," *2016 Dynamics of Systems, Mechanisms and Machines (Dynamics)*, Omsk, 2016, pp. 1-5.