Key Logger Biometric

Abstract:

As technology continuously advances with the wide use of distributing information through the internet, the more crucial it is to need digital security. Other than viruses and malware, becoming a victim to internet fraud can lead to stolen personal data and severely financially damages. To prevent this, authentication processes have been created. One type of authentication process is the use of a person's biometrics. Every person has a set of unique characteristics that distinctively identify themselves from others, such as fingerprints patterns, facial patterns, DNA pattern, and voice pattern. These biometrics authentications can also be applied to computer users' characteristics. This paper will look into the possibility of using a key logger as a biometric authentication process.

Background:

A user can verified themselves with either three types of authentication processes, knowledge base processes, object base processes, or through the use of biometrics processes. Knowledge base processes require the user to input some sort of answer to a question he/she may only know. Object based processes require the user to have some sort of physical key, such as a password token or a key. The last process, biometric based processes, can be broken down into two parts, physical or behavioral characteristics.

Biometrics authentication determines the identity of the user by their physical or behavioral characteristics. Physical characteristics can include verifying a user's fingerprints, iris/ retinal, or DNA. Behavioral characteristics can include verifying a user's handwriting, voice pattern, or keystrokes dynamics from a keyboard. Of the latter, keystrokes dynamics can be determined from characteristics such as typing speed, pushing key down times, flight times to next key, and two or three letter digraphs key times. To implement the behavioral keystroke authentication process, a key logger biometric will be further investigated.

Design:

To implement the Key Logger biometric, a GUI was created to actively interact with the user, see Figure 1 and 2. The GUI would first ask the user for their name so that the profile data may be saved under the user's name. The GUI would then illustrate a passage to the user. The user is then expected to type out the passage. As the user types, the GUI would actively highlight the typed text in either two colors, red or green. If the user's typed text letter did not match the GUI's sample text letter, the text letter would highlight red. If the texts matched, the text would highlight green.

The GUI used two sample passages. A passage from Huckleberry Finn was used as the training text dataset and a passage from wikipedia about Zebras was used as a testing text dataset. These passages were selected after research was done on the commonly used letters of the English language. Figure 3's table illustrated the percentage usage of each letter of the alphabet. The samples texts, Huckleberry and Zebra, appeared to matched the English language frequency percentages. Figure 4 illustrates the frequencies of the sample texts vs the expected frequencies of the English language.

The texts chosen were the result of surveying a number of websites that tested users typing speed. Originally, a collection of pangram sentences were considered, but the decision was made to choose length of text over letter coverage. In addition, excerpts from Huckleberry Finn were chosen as the initial training set because they appeared on multiple testing websites. This was freely available online [1]. The text about zebras appeared on http://www.typingtest.com/ which had used many uncommon letters such as 'z' and 'x'.

After each user typed both passages, each test run provided a training and testing dataset. With the Huckleberry Finn training and Zebra test datasets, the data was “cleaned up” for classification process. The classification process should be able to identify which Zebra test dataset match with the correct, the same user, Huckleberry training dataset. To do this, a Key Logger Extractor was created to examine the dataset’s digraphs, statistical characteristics, and outlier removal.

Once obtaining key log data, the next step was to perform feature extraction. This feature extractor was written in C# to remain consistent with the programming language that the logger was written in. A variety of features were calculated to provide an in depth classification for each user. These features were extracted from a log file that was created using a training text file and then compared to features that were extracted from a test text file. The features being extracted were average, minimum, maximum, and standard deviation of:

- Single key down time
- Single key fly time
- Digraph down time

The down time is the amount of time a key was pressed down until the same key was released. The fly time is the amount of time from any key being released until the key cared about is pressed. A single key feature considered any key press while a digraph considered two consecutive keys. The single key values ranged from 1 to 127 which is any possible key on a keyboard. The digraph values were only calculated if it matched one of the 12 most common digraphs in the English language.

Since the log file always contained two records for a single key press (KEY_UP, KEY_DOWN), it was safe to consider the first occurrence of a key code to be down, and the second to be up. Also, because multiple keys could be pressed at once, it could not be assumed that if a key was pressed that the next record in the log was that same key being released. To handle this issue, a queue was designed using a Dictionary where the key was the key code and the value was the timestamp. The following pseudo code explains the process of determining single key calculations.

```
Read log until EOF
    If queue.contains(key)
        Long time = key.time – queue.time
        //Perform necessary calculations
        Queue.remove(key)
    Else
        Queue.add(key, time)
```

When a key was found to be in the queue, it was assumed to be up and the time calculations were then performed. The final step was to remove the key from the queue to allow for the key to be pressed again.

In order to extract digraph information a buffer held the previously two typed keys. The digraph fly time was calculated by subtracting the release time of the second key from the down time of the first key. The only set of key pairs that we were concerned with was the 12 most common English digraphs ("TH", "HE", "IN", "EN", "NT", "RE", "ER", "AN", "TI", "ES", "ON", "AT"). Limiting the digraph set to the 12 most common gave us a more reasonable set of data to work with. If we were to implement all possible digraphs, the data for many of the digraphs would have been scarce, and therefore not reliable.

The output of the extractor went to a single file for use in Weka. The name of the input files and output file were provided in the command line. The usage is as follows:
KeyLoggerFeatureExtractor.exe <input files> <output file>
<input files> = log files (any number of log files >= 1)
<output file> = the name of the output file

By providing multiple input files, the output file could be submitted straight to Weka for evaluation based on Weka's specifications. This is because the output file contained an example row and a row for each input file. The output contained a value for each key code on a single row that matched the example row. If a 0 appears on a log row, it means the key was never pressed by the user. This format appears as follows:

```
Name,1,2,3,4,5...127
LogFile,0,12.54,93.39,12.48… 0
```

After examining the results from the first implementation of the feature extractor it could be seen that many of the averages for times were thrown off because of outliers. In order to fix this we cleaned the data by removing outliers. Outliers were selected by running the feature extractor function on a given file once to get the standard deviation for each single key fly time, single key down time, and digraph fly time. The input file was then run through the feature extractor a second time, omitting all times that had a deviation of more than twice the standard deviation. Cleaning the data gave an output file that more closely represented that average typing speed of a given user.

Analysis - Classification process:

Method 1 - Unweighted Nearest Neighbor

Initial research conducted when examining the possible feature sets that could be extracted had positive results on a customized nearest neighbor classifier that ran on the euclidean distances between particular features in their feature set [4]. For an initial implementation of this approach, an unweighted algorithm was used. Training data and testing data are pruned such that only features evident in both sets are considered. Each feature is considered as another dimension to which the overall distance is calculated. The results below show the various levels of success in matching the testing set (Nate Smith's Zebra testing dataset, User A) to the training set (containing Nate Smith's Huckleberry training dataset, user A). Data abnormalities were pruned prior to testing. See Figure 5.

Average "press time" per keystroke was the only metric used per feature. The dataset defined as "Single Keys" include all keys registered between both data sets (including shift, special non-alphanumeric keys, etc) as features for classification. The dataset defined as "Digraphs" include only common digraphs listed previously as features for classification. The dataset defined as "Full" contain all features that were common between the training and testing dataset. As is apparent in the table above using euclidean distance with unweighted dimensions is not largely accurate.

When using the full feature set with this method, the testing data is incorrectly classified with a ∆*intended distance to classified distance* ratio of (1202.9 - 985.5) / 985.5 = 0.221, which is somewhat poor. Examining the single key error rate in the same manner yields (820.3 - 665.4) / 665.4 = 0.233, which is even more inaccurate. This same classification, however, correctly classified the testing data when looking only at the digraph patterns.

This same unweighted approach was also used to examine the average flight time of these same keys, as well as a combination of press and flight time. Because we only extract the average downtime for digraphs, they were included in all feature sets. The findings are shown in Figure 6.

As is apparent in the table, the inclusion of average flight times between keystrokes alone does not solve the accuracy issue. The ∆*intended distance to classified distance* ratio for the average key press time is (1202.9 - 985.5) / 985.5 = 0.221, which is more accurate than this same metric using average flight time features (897.2 - 708.1) / 708.1 = 0.267. Using both types of features together proved to be the most accurate, though it still classified incorrectly. The is ∆*intended distance to classified distance* is calculated to be (1226.6 - 1009.1) / 1009.1 = 0.216.

Method 2 - Weighted Nearest Neighbor Classification Using Relative Key Press Timings

The goal of this classifier was to use a feature that would be independent of typings speed. Classifying based only on average key timings can be problematic because It requires that the user types at a consistent speed not only throughout a single sessions, but through all later sessions as well. Relative key timings seek to minimize the impact of changes in typing speed and extract features that are most unique to individual typists. For this paper, the general definition of a relative key press time is the length of time from when a key is pressed to when it is released (the press time) divided by the current average press time. The average relative key press time for a specific key is the average of all relative key press times for that key.

As implemented an unweighted rolling average is kept for the past 50 key presses regardless of which keys are pressed. While the average is being initialized, no relative times are calculated and no data cleaning is performed. This limits the amount of bias that is introduced to the by the initial key timings to the final average relative key press timings.

Once the average is initialized, some data cleaning is performed on all later key presses. Specifically, any key press lengths that are shorter than an eighth of the current average or longer than eight times the current average are ignored. The scale factor of eight was chosen to include the majority of key presses including shift (which is typically longer than other keys) while excluding outliers. Outliers include key press lengths of zero (due to a timing bug) or extremely long key presses such as holding backspace or shift to affect multiple characters.

For each key press that is not an outlier, the relative key press time is included in the unweighted average of relative key press times for the specific key and a count for the number of times the key was pressed is incremented. This is done for all remaining key presses in the log. The final result is a profile containing average relative key press times for all keys that were pressed during the log session.

During classification, each input profile is compared against all known profiles and the best match is selected. The matching algorithm is a modification on a nearest neighbor classifier in which different features have a weighted contribution to the final distance value. The features used for matching are the average relative key press times. Only features that are common to both profiles being compared are used. The weight for each feature is the total number of times the respective key was pressed in both profiles. This weights frequently pressed keys more heavily than less frequently pressed keys. The distance for individual features is the magnitude of the difference between the average relative key press times in each profile multiplied by the weight as described. The overall distance between two profiles is simply the sum of all feature distances divided by the number of features common to both profiles. The division is necessary to avoid favoring profiles with few matching features. In order to make the comparison more reliable features that do not meet a minimum number of key presses in both profiles are discarded. For this implementation, the minimum was set to two.

The results are quite promising but the size of the training and test sets was limited. The training set consists of profiles for twelve unique users typing a passage from Huckleberry Finn. The test set consists of profiles for five unique users typing a passage derived from a few wikipedia articles. The five users included in the test set are a subset of the twelve users included in the training set. All five of the test samples were classified correctly although only four of the five were classified with good separation from the other samples. One of the test samples was a relatively close match to two of the training profiles but had good separation from the rest of the profiles.

Weighted city block distance as used in the classifier:

$$d = \sum_{n} |t_{ia} - t_{ib}| \cdot (c_{ia} + c_{ib})$$

Where t is the average relative press time for a key and c is the number of times the key was pressed.

The results are shown in Figure 7. For simplicity, in the chart above, the test and training sets have been relabeled with single letters corresponding to each unique user. The five users common to the training and test sets are “F”, “G”, “H”, “J”, and “K”.

Method 3 - <u>Classification with NN and MATLAB:</u>

The software package MATLAB offers a statistics toolbox for computing the nearest neighbor using a variety of different distance functions. I thought it would be interesting to survey these different distances and see which ones, if any, outperform the others.

The function knnsearch(*X*,*Y*) finds the nearest neighbor of *Y* in *X*. It was relatively straightforward to load the Huckleberry data as *X* and the zebra data as *Y*. Ryan and

Troy's extractor function was used with the average hit and flight times for all keys, as well as the digraph data. I used the modified code that removes outliers. Similarly to Weka, MATLAB can load CSV files, and automatically stores them as a matrix. The data was trimmed by removing all columns *j* from *X* and *Y* if the *j*-th column contained only zeros in both. The number of columns went from 266 down to 117.

Seven different distance functions were tried. The distance is denoted by *d* and *x* and *y* rows from *X* and *Y*. In this case, they correspond to tests from the Huckleberry and zebra texts. The functions are listed in Figure 9.

Notice that *City Block* is a special case of *Minkowski* when *p*=1. Likewise, *Euclidian* is a special case of *Minkowski* when *p*=2. I chose *p*=5 for these tests. From Figure 8 display the results from this method. For some reason or another, "Alex", "anon17" and "jfdihdjh" made almost every match. This might be caused from not cleaning the dataset well enough.

Method 4 - <u>Machine Learning with WEKA:</u>

WEKA is an Open-Source machine learning suited provided by the University of Waikato [5]. Utilizing WEKA, an attempt was made at using built-in classifiers to provide user identification and matching. Although WEKA provides a wide variety of machine learning algorithms, only one was found to be suitable for the application of identifying the uniqueness of individual users.

In WEKA, the first task in creating a database for comparison is through a "training set". Since every user will be unique, it follows that a unique classification must be produced from the training set. In other words, one user's keystrokes must be classified as one and only one user. Of all the classification algorithms used in WEKA, only "LADtree" was found to maintain the requirement of a one-to-one classification.

With the classifier determined, the next step is to attempt matching users with a text that is not part of the training set. While the training set was built on an except from

Huckleberry Finn, the test set is created from the body of the Wikipedia article on zebras.

The preliminary results were rather poor and had an accuracy of only 20%. However, it was observed in the training set that certain keystrokes occur infrequently and could potentially be removed to improve accuracy. The viability of this simplification can be seen in Figure 4 as several characters such as ‘j’ or ‘z’ occur relatively less frequently in the sample texts than, for example, ‘e’ or ‘i’. With this simplification in place, the matching accuracy improved to 60%.

Conclusion:

Before looking into implementing the key logger biometric, one may wonder if there are any advantages to using this process. The major benefit to the key logger biometric is that it can be implemented with little hardware, a commonly used keyboard, since the process is fundamentally software. This makes the process cost effective and easily distributable while providing computer/internet security. However this process does have its disadvantage.

The main drawback of the key logger biometric is the need for the user to be consistent. Computer users may face several scenarios of fatigue, change of dynamic typing patterns due to improvement and/or deterioration in typing, injuries, change of hardware device (different keyboard). Since the process is mainly software based, there are many possible methods to overcome these drawbacks. These challenges can be overcome with features and artificial intelligence. Some features being investigated are the use of a long fixed training text, the use of digraphs, the use of statistics, outlier removal, and AI classification tools.

From method 1, the results indicate two things. Firstly, unweighted euclidean distance does not accurately represent the typing pattern as a whole. One possible idea to making this sort of model more accurate is to have features weighted in terms of “importance”, which can be defined by the frequency of the keystroke. Secondly, these results also suggest that average latencies over digraphs latencies offer a more unique manner of identification.

From method 2, by cleaning up the datasets and removing outliers of 0.125 - 8 pressed lengths the classification process was more effective than method 1. It was able to accurately classify five of five users' datasets. The idea of using weights to the classification process was also examined. By counting the number of time the key was used, and disregard the less used letters, the classification process would be able to use useful features instead of fallacious features.

With an increase in matching accuracy after the removal of infrequent keystrokes, it follows that for machine learning to be an applicable user identifier, some data "pruning" may be necessary. From the results of method 4, it would appear that the over-provision of information introduces errors that can be corrected if less-relevant information is removed from the training set. What is not implicitly clear is what kind of data removal is necessary for better matching. What attributes are *consistent across* multiple users and what attributes are *distributed*? Further quantitative testing on the efficacy of specific features for accurate matching will need to be done before removing additional data.

Results

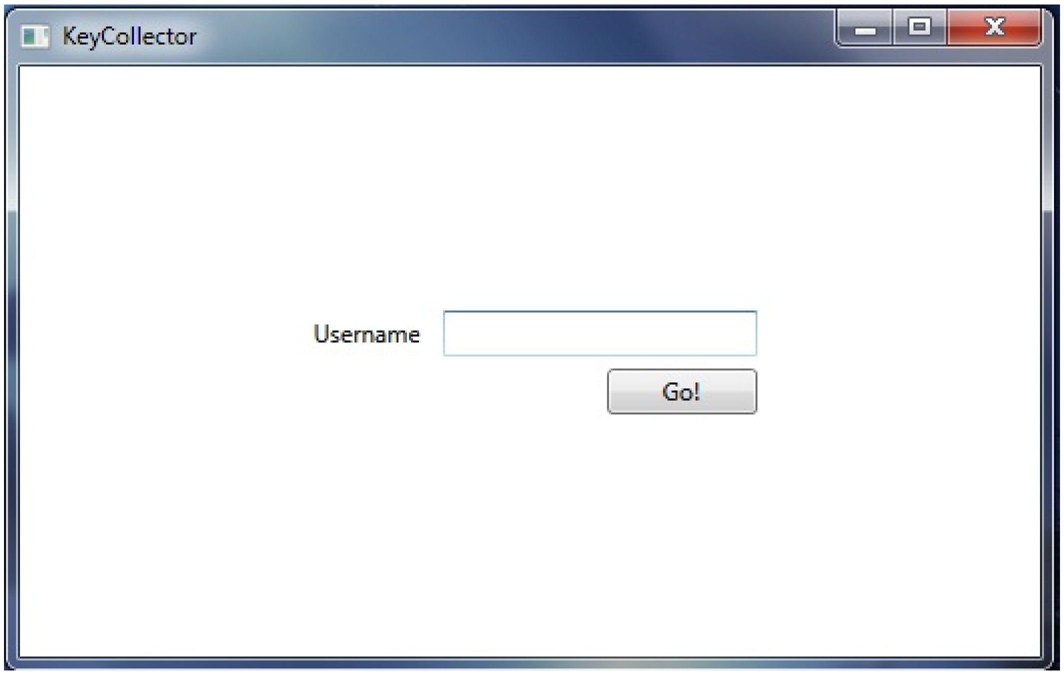
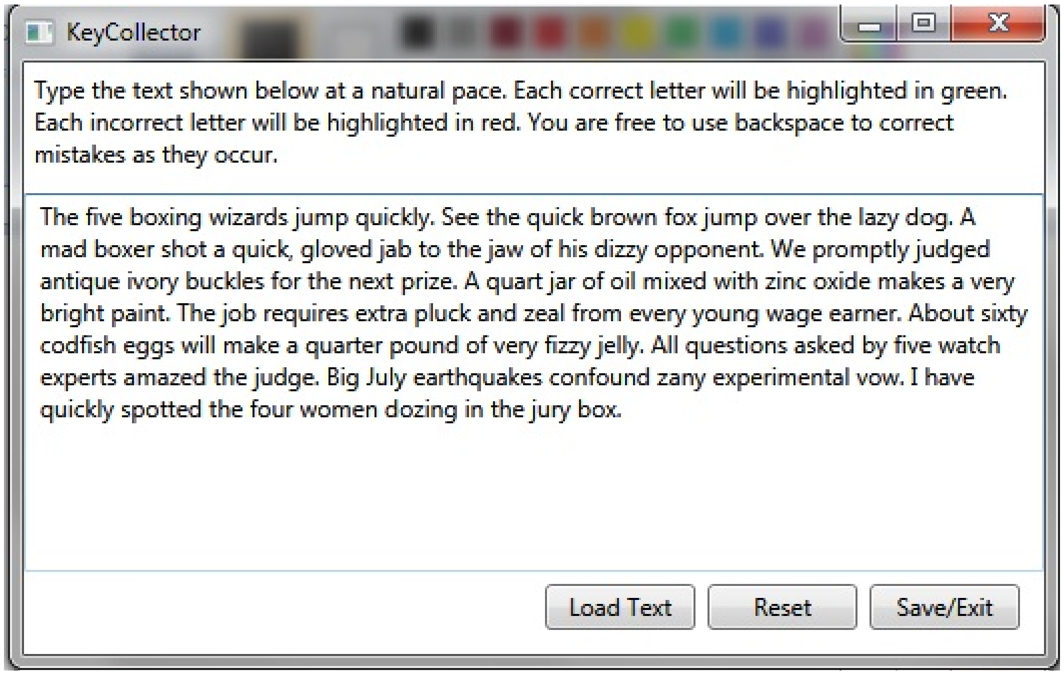


Figure 1: KeyCollector's Login Window



Type the text shown below at a natural pace. Each correct letter will be highlighted in green. Each incorrect letter will be highlighted in red. You are free to use backspace to correct mistakes as they occur.

The five boxing wizards jump quickly. See the quick brown fox jump over the lazy dog. A mad boxer shot a quick, gloved jab to the jaw of his dizzy opponent. We promptly judged antique ivory buckles for the next prize. A quart jar of oil mixed with zinc oxide makes a very bright paint. The job requires extra pluck and zeal from every young wage earner. About sixty codfish eggs will make a quarter pound of very fizzy jelly. All questions asked by five watch experts amazed the judge. Big July earthquakes confound zany experimental vow. I have quickly spotted the four women dozing in the jury box.

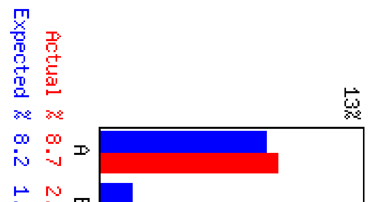
Figure 2: KeyCollector's Typing Window

	Expected	huckleberry.txt	zebras.txt
a	8.17	8.6	9.0
b	1.49	2.1	2.0
c	2.78	1.7	2.7
d	4.25	5.7	4.2
e	12.70	9.9	11.6
f	2.22	1.1	2.1
g	2.02	2.3	2.2
h	6.09	5.7	5.1
i	6.97	5.9	6.5
j	0.15	0.2	0.1
k	0.77	1.0	0.8
l	4.03	4.1	4.6
m	2.41	2.5	2.6
n	6.75	6.4	6.5
o	7.51	8.4	7.6
p	1.93	1.2	1.8
q	0.095	0.0	0.2
r	5.99	4.0	5.6
s	6.33	5.4	6.6
t	9.06	9.8	8.7
u	2.76	3.5	3.4
v	0.98	0.5	1.0
w	2.36	3.5	2.0
x	0.15	0.1	0.6
y	1.97	2.0	2.1
z	0.074	0.0	0.4

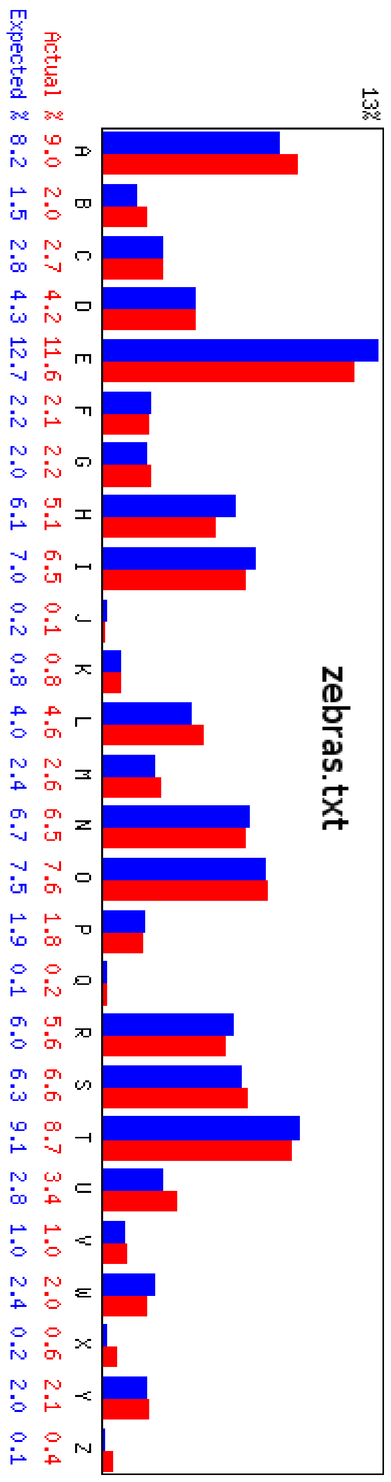
Figure 3: Frequencies of English Letters Table in Percentages [2][3]

Figure 4: Frequencies of English Letters vs Huckleberry and Zebra Sample Text [3]

```
HUCKLEBERRY FINN EXCERPT:
Common Digraph Coverage: 26/27 96%
Common Trigraph Coverage: 8/13 61%
Common Doubles Coverage Coverage: 7/7 100%
Full Digraph Coverage:244/676 36%
```



```
ZEBRA WIKI PAGE:
Common Digraph Coverage: 27/27 100%
Common Trigraph Coverage: 10/13 76%
Common Doubles Coverage: 7/7 100%
Full Digraph Coverage: 277/676 40%
```



Euclidean Distances for Nearest Neighbor Classification with Unweighted Features

	User A	User B	User C	User D	User E	User F	User G
Full	1202.9	1751.4	1143.6	1167.8	1125.4	1076.5	**985.5**
Single Keys	820.3	1066.4	912.4	772.1	715.0	682.7	**665.4**
Digraphs	**251.6**	578.0	409.2	374.4	344.0	307.1	466.5

Figure 5: Method 1 Unweighted Nearest Neighbor classification of Nate (User A) Zebra testing dataset to other Users' Huckleberry Training dataset.

	User A	User B	User C	User D	User E	User F	User G
Down Avg	1202.9	1751.4	1143.6	1167.8	1125.4	1076.5	**985.5**
Fly Avg	897.2	1410.2	**708.1**	894.2	886.1	847.8	743.5
Both Avg	1226.6	1780.3	1165.2	1192.9	1150.0	1099.3	**1009.1**

Figure 6: Method 1 Unweighted Nearest Neighbor classification of Nate (User A) Zebra testing dataset to other Users' Huckleberry Training dataset.

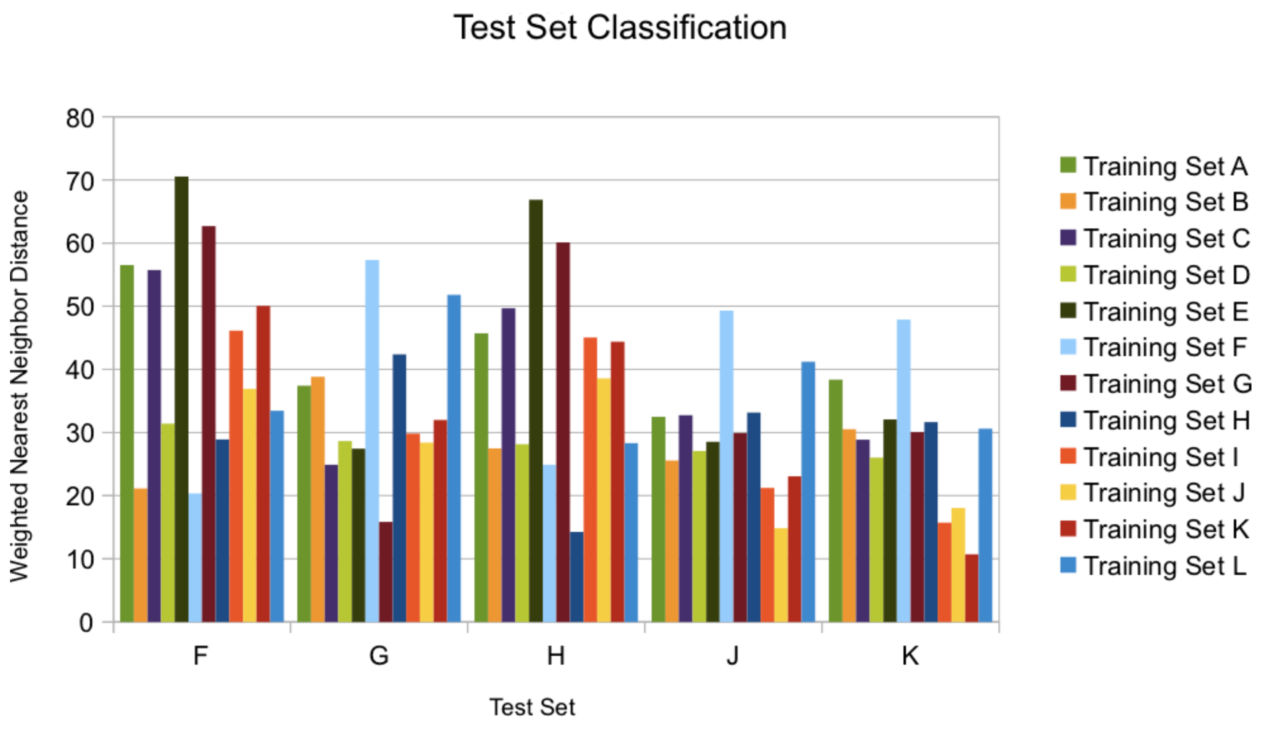


Figure 7: Method 2 Weighted Nearest Neighbor Classification Using Relative Key Press Timings. Comparing Zebra Testing dataset to Huckleberry Training datasets of different Users (F-K)

User	Euclid	Std Eu	City	Mink	Cheby	Cosine	Corr.
Alex	Alex	anon17	Alex	Alex	Alex	anon17	Alex
Joe	Alex	anon17	Alex	anon17	anon17	Alex	Alex
Nate	anon17	anon17	anon17	anon17	anon17	anon17	Alex
Sarah	Alex	anon17	anon17	anon17	anon17	anon17	Alex
Tylar	jfdihdjh	anon17	Pete	jfdihdjh	Alex	jfdihd	Alex

Figure 8: Method 3 Classification with NN and MATLAB Five zebra tests were compared with 12 Huckleberry tests. The nearest neighbor found for each zebra test for each distance function. (Red texts are different functions, Black texts are different Zebra testing datasets, and Blue tests are different Huckleberry training datasets that were matched)

1. *Euclidean,*

$$d^2 = (x - y)(x - y)'$$

2. *Standardized Euclidean,*

$$d^2 = (x - y)V^{-1}(x - y)'$$

3. *City Block,*

$$d = \sum_{j=1}^{n} |x_j - y_j|$$

4. *Minkowski,*

$$d = \sqrt[p]{\sum_{j=1}^{n} |x_j - y_j|^p}$$

5. *Chebychev,*

$$d = \max_j \{|x_j - y_j|\}$$

6. *Cosine,*

$$d = 1 - \frac{xy'}{\sqrt{(xx')(yy')}}$$

7. *Correlation,*

$$d = 1 - \frac{(x_s - \overline{x})(y - \overline{y})'}{\sqrt{x - \overline{x})(x - \overline{x})'}\sqrt{(y - \overline{y})(y - \overline{y})'}}$$

Notice that *City Block* is a special case of *Minkowski* when *p*=1. Likewise, *Euclidian* is a special case of *Minkowski* when *p*=2. I chose *p*=5 for these tests.

Figure 9:List of functions tested for method 3

	A	B	C	D	E	F	G	H	I	J	K	L
A	X											
G						X						
H						X						
J									X			
L											X	

Figure 10: Matching results with the removal of keys that occur *less than* 10 times.
(Black texts are different Zebra testing datasets and Blue tests are different Huckleberry training datasets that were matched)

[1]	A	B	C	D	E	F	G	H	I	J	K	L
A	X											
G						X						
H								X				
J										X		
L										X		

Figure 11: Matching results with the removal of keys that occur *less than* 30 or 50 times.
(Black texts are different Zebra testing datasets and Blue tests are different Huckleberry training datasets that were matched)

[1] Identical results were obtained for 30 and 50.

References

[1] Twain, Mark, and Donald McKay. *The Adventures of Huckleberry Finn*. New York: Grosset & Dunlap, 1948. Accessed from the Electronic Text Center of University of Virginia Library at http://etext.virginia.edu/toc/modeng/public/Twa2Huc.html

[2] Robert Edward Lewand. Relative Frequencies of Letters in General English Plain Text. *Cryptographical Mathematics*. Accessed from http://pages.central.edu/emp/lintont/classes/spring01/cryptography/letterfreq.html

[3] Frequency Analysis. Thonky. Accessed at http://www.thonky.com/kryptos/frequency-analysis/

[4] Keystroke Biometric Recognition Studies on Long-Text Input under Ideal and Application-Oriented Conditions
http://www.csis.pace.edu/~ctappert/papers/cvpr2006.pdf

[5] WEKA-The University of Waikato. Accessed at http://www.cs.waikato.ac.nz/ml/weka/