An Exploration of Feature Extraction and Classification Techniques for Keyboard-Based Continuous Authentication Systems

Christopher Wood*†
*Department of Computer Science, †Department of Software Engineering,
Rochester Institute of Technology
Email: caw4567@rit.edu

Abstract

In this report we discuss the major components of a term-long project on keyboard-based biometric continuous authenticaton systems. The work was divided into two parts: designing, implementing, and testing an evaluation system that is based on dynamic free-form text input to perform identification, and analyzing a variety of feature extraction and data filtering techniques that can be utilized to help improve the identification accuracy of this system. Our experimental analysis is divided into the evolutions of the evaluaton system, and we discuss the benefits and drawbacks of each tested technique. We also document the results from a literature survey on the specific topic of identification systems based on dynamic free-form text input.

I. Introduction

Authentication has traditionally been a task performed during the sign-on phase of a computer system. However, there has been substantial research efforts in the past decade that focus on continuous authentication, which is a technique where a system is able to detect user changes during a given session. Due to the semi-autonomous nature of typing, user input captured through the keyboard provides the ability to create unique user biometrical profiles [1]. In other words, typing patterns are behavioral characteristics and thus are distinct between separate users, which is why they have been studied for use in continuous identification and authentication schemes [2]. Furthermore, since the keyboard is the most common input device for computer systems, this provides a low-cost and non-intrusive way to identify the host user.

Keyboard-based biometric systems typically come in two forms: fixed input and free-form input. Fixed input systems perform authentication by requiring a user to transcribe a selected passage, and the biometrical features extracted from the transcription process are compared against a known set of user profiles to determine their identity. These forms of systems tend to be more accurate because they limit the input alphabet and constrain the user to a specific goal while typing. It should be noted that, in this case, accuracy refers to the percentage of successful user identifications. In other words, if a system attempts to identify a user n times and only m of these samples are correct, then the accuracy for this system is $(n/m) \cdot 100$. These systems can easily be deployed in environments where no further form of text-entry is required after the initial authentication phase.

Free-form input systems involve continually (or periodically) monitoring keystroke typing patterns in order to establish the host user identity. We refer to these systems as dynamic free-form text input systems because of this periodic behavior based on any form of text input. Since the type of input and the user goals for using the system can vary widely when using these systems, the accuracy measures generally tend to be lower.

By definition, continuous authentication schemes have significant privacy implications. User-sensitive data that is entered through the keyboard must remain confidential. This requirement places constraints on how such biometric solutions persist and use keyboard data for user-profiles and classification. While implementation techniques such as feature database encryption help maintain the confidentiality of this

sensitive data, the more important implication of this requirement is that all features that are extracted must not embody any information related to what is being entered by the user. That is, passwords or other specific text entries should not be stored. Rather, features that encapsulate the typing patterns of a user (e.g. key press duration and key fly time) must be the only entities recorded as part of a user profile.

In addition to these security issues, there exists strong relationships between other non-technical factors, such as the environment and user fatigue, stress, and injury. These factors inevitably lead to systems that produce a relatively high number of false positives and negatives, as they are outside the scope of data that can be collected from the keyboard and thus introduce noise into the dataset [3]. Therefore, in this work, we make the assumption that the user is typically in the proper physical condition necessary to use the system "normally".

In this paper we first analyze the state of the art of continuous authentication schemes. We then report on a term-long project in which fixed-string classification techniques were evolved to build an identification system based on dynamic, free-form input. This system extracts features from keystoke data that is collected in real-time to generate a user profile, which is then compared against a known set of profiles to determine the host user's identity. We discuss the feature extraction and classification techniques used by this system in detail. We also present a novel perspective of research in this area consisting of a typing pattern analysis of users in different contexts (or with different situational goals). The information we gained from this analysis is then incorporated into the feature extraction and classification techniques to improve the performance of identification systems based on dynamic, free-form input. Finally, we conclude with a thorough discussion of the experiments conducted within the scope of our evaluation system and with the help of WEKA, a data mining and machine learning tool suite [4].

II. Related Work on Dynamic Free-Form Text Input Systems

Continuous authentication systems based on keyboard-based biometric authentication systems is not a new research topic. Original publications related to the area date back to the 1970s, with pioneer work conducted by Spillane [5]. Since then, significant research has been done on continuous authentication schemes based on dynamic free-form keyboard input with different threads of research focusing on feature extraction and classification techniques.

Monrose and Rubin [6] present some of the original work in free-form input systems. They experimented with a total of 31 typing data sets collected from transcriptions of well-defined phrases and some unknown sentences. From this data, the authors formed user profiles that consisted of standard data such as the mean and standard deviation of digraph fly times and keystoke press durations. To improve their results, the authors trimmed outliers from the dataset by removing all feature values that were T-standard deviations away from the mean for a given feature. Unfortunately, their approach yielded significant outlier removal and led to a user profiles with smaller dimensions and less information. In fact, when $T = 0.5$ the authors reportedly discarded approximately 50% of their data set.

Since user profiles were represented as n-dimensional vectors of feature values, Monrose and Rubin experimented with the standard Euclidean distance classification approach (i.e. classification is based on the smallest distance between the test profile vector and trained user profile vectors). While this approach led to high identification accuracy when both the trained and test data were generated from fixed (or structured) input, the accuracy fell to approximately 23% when the test data was based off of free-form input.

Dowland et al. present a more novel approach based on adaptive and weighted user profiles [7]. In their work, user profiles were built over a span of four weeks from normal user behavior. As a result, the user input was completely unconstrained. However, user profiles were based entirely on digraph timing features, and due to the length of the training period, they usually contained thousands of digraph samples. Feature weights came into play when the system was determining whether or not to "accept" new digraph features for the user model U. To be more specific, U marked a new digraph measure D as "accepted" if and only if the value fell between the interval $[D_e^p \pm w \cdot D_\sigma^p]$, where D_e^p and D_σ^p are the mean and

standard deviation values of profile p for digraph D. In this way, we see that the outlier removal was variable and based entirely on the weights assigned to each vector. The derivation of these weights is not discussed, but clearly is an important factor that contributed to the reported classification accuracy. Another difference between the work by Dowland et al. and Monrose and Rubin is that classification was determined by finding the user profile that yielded the largest number of "accepted" digraphs. This technique enabled the weights to account for the degree of change in a user's typing patterns, which is a major problem that most free-form input systems face.

III. Evaluation System

Keyboard-based biometric authentication and identification systems that are based on dynamic, free-form input consist of two main components: a feature extraction and classification module. In order to satisfy both the usability and security requirements for keyboard-based authentication schemes, these components must be transparent to the user. That is, the data acquisition module should not have noticeable impacts on normal usability, and the feature extraction and classification modules should not monopolize system resources during run-time. In order to experiment with new feature extraction and classification techniques that can be used to improve the accuracy of dynamic free-form text input identification, we have designed and implemented a relatively unobtrusive dynamic identification system that runs on Unix platforms. Figure 1 depicts the runtime deployment diagram of the system and shows the mapping of the three aforementioned modules onto runtime daemon processes that work together to perform user identification. The static software of this system is shown in Figure 2. As you can see, there is a clear flow of data from the `Keylogger`, which is implemented in C in our evaluation system, to the `Classifier`, which is part of the Python code base.

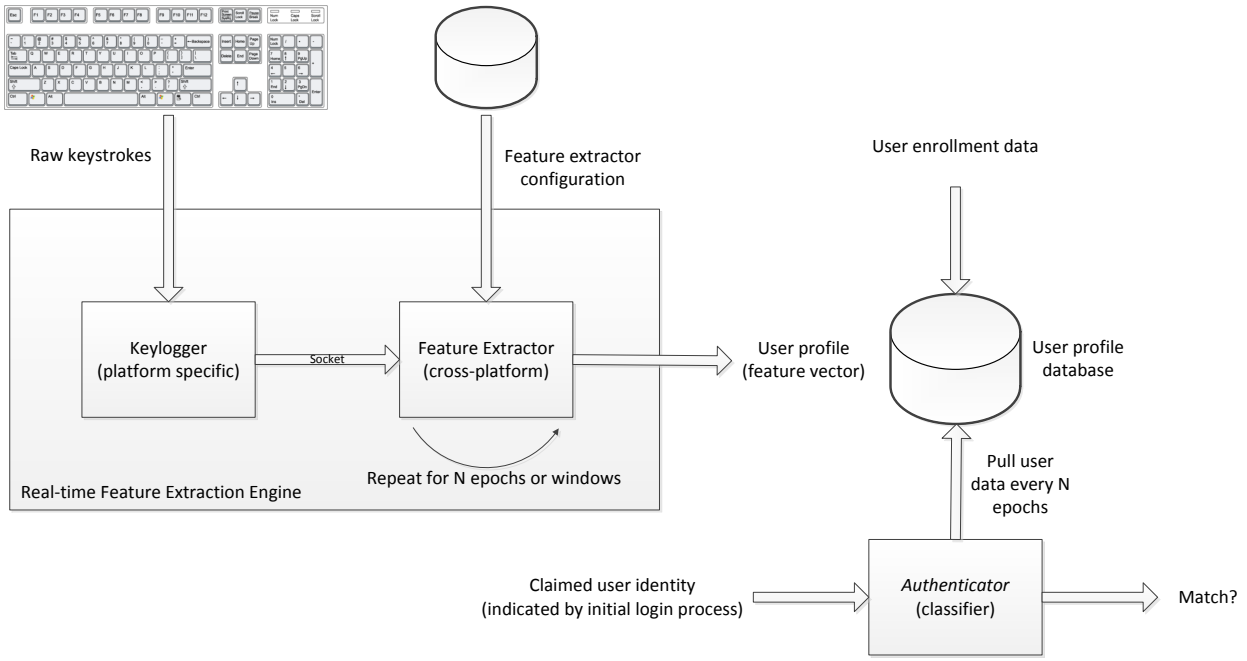

Fig. 1. The runtime cooperation diagram for the evaluation system. The `Keylogger` is platform specific and the rest of the system is independent of the underlying operating system, which makes our system very portable. We use sockets to transfer data from the `Keylogger` to the `Feature Extractor`, which means that these two processes need not run on the same host. Although this was not tested, it is expected that running these on two separate systems would impact the system performance.

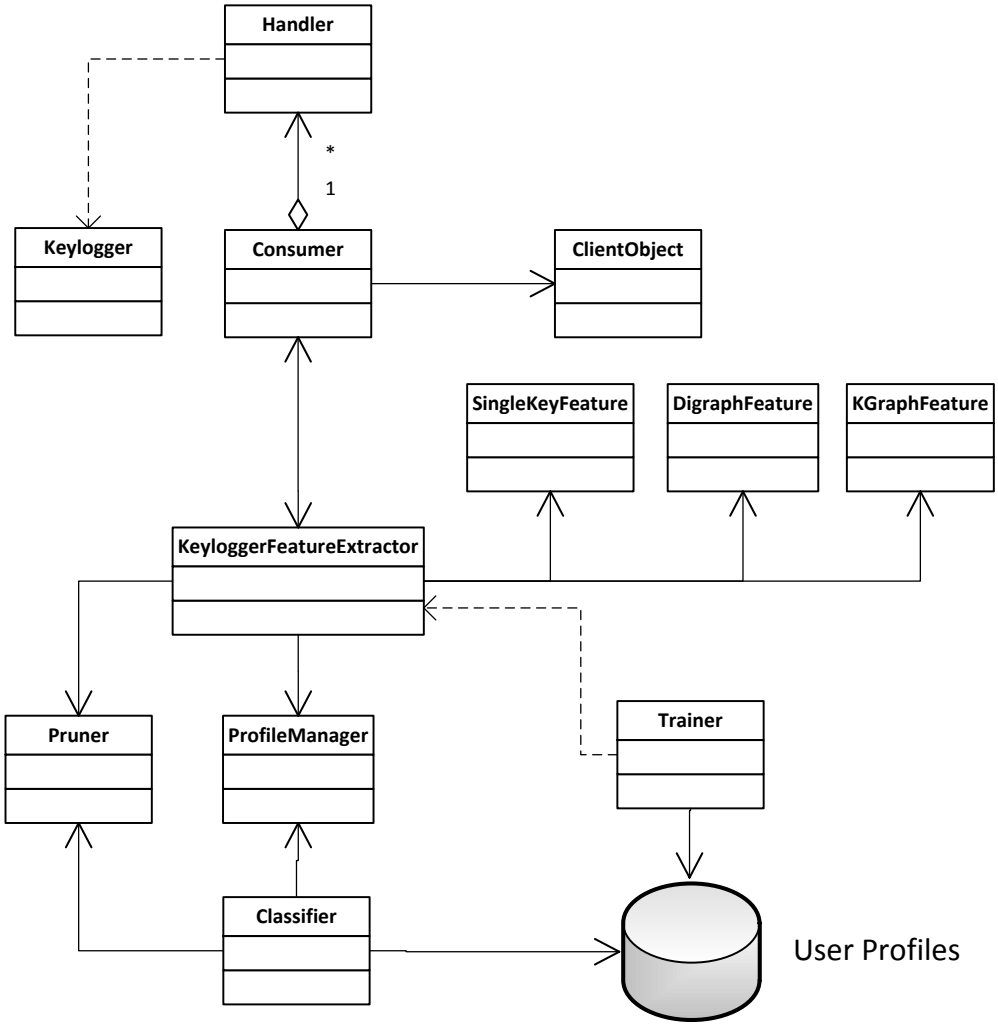

Fig. 2. The static software structure for the evaluation system. The user profile is currently stored as a flat CSV file. For security purposes in a real deployment, this would need to be a protected database with the appropriate access controls to prevent malicious users from identifying keystroke typing patterns for users and then exploiting this knowledge for system misuse.

At a high level, the sequence of system activities that make up the identification process is as follows:

1) The `Consumer` establishes a connection with the `Keylogger` and spawns a `Handler` thread to read all data from the recently created socket.
2) The `Handler` forwards all data to the `Consumer`'s internal queue.
3) When the `KeyloggerFeatureExtactor` time limit expires or the `Consumer`'s queue is filled to the brim the data is removed from the queue, appropriate features are extracted from the data, and a user session profile is created and sent to the `ProfileManager` for storage.
4) When the `Classifier` reaches its time limit, it queries the `ProfileManager` for all of its stored user profiles, merges them together by performing statistical averages across all features in the profiles, and then attempts classification against the profiles stored in the user profile database (i.e. the one constructed after training the system).

The `Trainer` is a helpful program that executes outside of the the context of the evaluation system to read user keystroke log files and build the user profile database. Alternatively, functionality exists to export the user profiles to an ARFF file for evaluation within WEKA. Several other special-purpose programs were implemented to aid in the system evaluation process. Their details are contained within the code documentation.

The system is very straight-forward to use. One must first start the `Consumer` process in the Python code base, as shown in Figure 3. When this is complete, the system will initialize the `KeyboardFeatureExtract Handler`, and `Classifer` threads to handle all incoming keylogger connections. The startup menu is shown in Figure 4.

```
/src/extractor$ python Consumer.py
```

Fig. 3. Initialization of the `Consumer` process in the Python code base.

```
Starting keylogger consumer. Building the thread components.
Consumer created.
Beginning extractor loop.
Classifier is starting to sleep
 Handler created.
Beginning handler loop.
-----------------------------------
Type 'help' or '?' for available commands.
-----------------------------------
>>
```

Fig. 4. The startup banner and menu for the Python (classification) portion of the evaluation system.

After this is complete, one must start the keylogger process and point it to the location of the `Consumer`, as shown in Figure 5. As you can see, superuser privileges must be granted to the keylogger, as it requires direct access to the keyboard status and port MMIO registers in order to read keystroke information from the user. If the `Consumer` address information is entered correctly, the two processes will establish a connection, as shown in Figure 6.

```
/src/keylogger$ sudo ./keylogger localhost 9998
```

Fig. 5. Initialization of the keylogger.

```
>> Client connected from ('127.0.0.1', 55232).
received: 28 KEY_DOWN
received: 28 KEY_UPN
received: 31 KEY_DOWN
received: 31 KEY_UPN
received: 20 KEY_DOWN
```

Fig. 6. The connection information that is displayed by the `Consumer` process.

At this point, assuming the system has been trained using existing keystroke log files and the `Trainer` script, the system can begin collecting keystroke information from the user in the background. On the test machine, the keystroke collection mechanism was unobtrusive and did not impact user typing patterns. A visual depiction of the system being used at runtime is shown in Figure 7.

In terms of performance, our evaluation system is a bit more resource intensive than we would have liked. The feature extraction and classification components in the Python code base consume approximately 70%

of the system CPU resources and 2.4MB of memory. The keylogger in the C code base uses approximately 25% of the system CPU resources and less than 1KB of memory. In terms of CPU performance, the socket communication contributes to the majority of the consumed resources. Due to the Python socket API, the `Handler` thread that reads data from the keylogger socket must poll the channel until data is available, which means that it never sleeps. This could be avoided by forcing timed sleep events after every polling event, but this would sacrifice the real-time performance of the system. We plan to explore this issue in greater detail in future work.

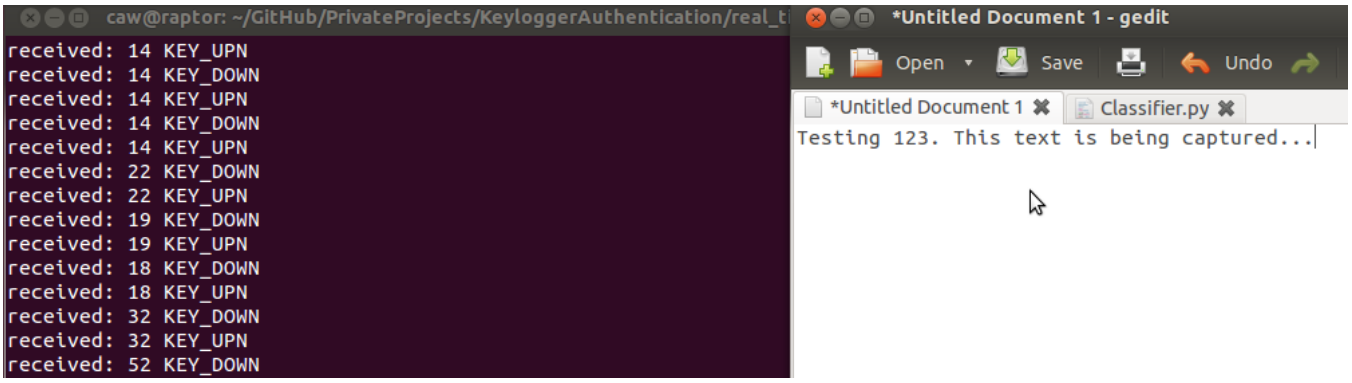

Fig. 7. Visual depiction of the keystroke information being gathered by the `Consumer` process while the user is typing in gedit, a popular text editor.

We now describe some more technical details on the system internal components in the following sections.

A. *Feature Extraction*

User profiles are based entirely upon the features that can be captured from the keyboard at run-time. Ever since the landmark report by the National Science Foundation and the National Bureau of Standards in the United States in the late 1980s there have been many different proposed features that can be extracted from a user's interaction with a system. For brevity, we present our final set of features in Table I, and refer to all of these features as the feature set $\mathcal{F}$. Figure 8 shows a visual depiction of how these features relate to keystroke information.

TABLE I
SELECTED FEATURES EXTRACTED FROM KEYBOARD BIOMETRIC DATA. THIS DATA ENCOMPASSES THE ENTIRE FEATURE SET AS USED IN THE FOURTH EVOLUTION OF OUR SYSTEM (SEE SECTION IV FOR MORE INFORMATION).

Feature ID	Description	Weight
1	Single key press duration (average)	1.0
2	Single key press duration (standard deviation)	1.0
3	Single key fly time (average)	1.0
4	Single key fly time (standard deviation)	1.0
5	Digraph time (average)	1.0
6	Digraph time (standard deviation)	1.0
7	Relative typing speed	1.0
8	Relative error frequency	1.0

It was shown by Robinson et al. [8] that key-hold duration times are more statistically significant than fly-times. Therefore, if we were to define a specialized weight function for the feature set $\mathcal{F}$, we would give priority to these specific features over the rest. Due to time constraints, we were not able to explore this issue in great detail.

Secondary or auxiliary features can also be derived from these measurements. For instance, the minimum and maximum typing velocity, in addition to standard statistical calculations on the dataset (e.g. mean and

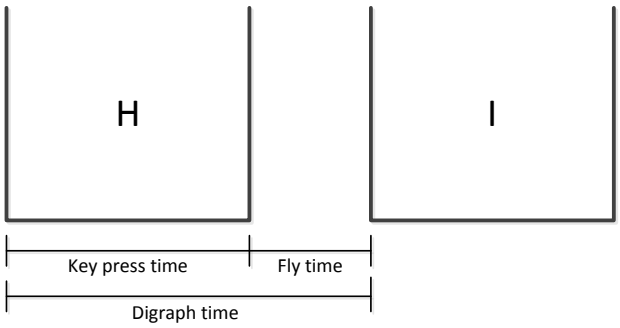

Fig. 8. Visual depiction of the timing features collected from the keystroke log files. In this image the user typed the phrase "HI".

standard deviation), can be combined into a user's profile in order to make classification more accurate. We may also consider certain typing practices when building a user profile, such as the average number of spaces entered after a period and comma usage in regular language. These measurements, however, may be sensitive to the type of data that is being entered. For example, software developers and journalists enter very different types of data into a computer using a keyboard. Developers write code, which does not have the grammatical structure or style of natural language, whereas journalists write articles that do have these properties. To account for these differences, however, user occupations and habits could be stored in user profiles to help classification.

Other measurements have been proposed in the literature, such as the total duration required to enter a specified string or passage and the amount of pressure applied to individual keys while typing. However, select string matching techniques are mainly applicable to static data acquisition techniques, and pressure-sensitive keyboards are not common among users. Thus, for the creation of user profiles, we limit ourselves to the measurements shown in Table I, and briefly experimented with some secondary features that can be derived from them.

Once extracted, the features are stored in a user profile U, where U is simply a vector of n features (i.e. $U = \mathcal{F}^n$). For our system, all timing information was recorded on the scale of milliseconds. Smaller precision did not seem to benefit the identification accuracy.

B. Data Filtering

Since our system is based upon free-form text input there is usually a significant number of feature outliers that can be removed in order to yield greater differentiation between separate user profiles. The most standard data filtering techniques are those based on the statistical properties of the user profiles. That is, outliers can be determined by identifying all features that are d-standard deviations away from the mean for a given feature. However, instead of determining outlier trimming based on individual features, we propose to use collections of features to determine our mean and standard deviation values. For example, since our user profiles consist of keystroke press durations for all available keyboard keys, we treat this as a single feature "group" $fg \subset \mathcal{F}$ and calculate the mean μ_{fg} and standard deviation σ_{fg} for keystroke press duration features across all features in this group. Then, outlier removal for features in this group is done by trimming all features whose value falls outside of the interval $[\mu_{fg} \pm d \cdot \sigma_{fg}]$. Algorithm 1 shows the technical details for this filtering operation. This filtering technique is performed during user training periods and when the system is running.

We also make several improvements to the generation of user profiles based on the input data acquired from the user. In particular, we recognize that, as the number of samples for a given feature increases, the statistical stability of the median value for the acquired samples surpasses the stability of the true mean value. As a result, our data filtering and user profile generation process provides a boundary k for which

Algorithm 1 Feature group statistical outlier removal

Require: A set of user profiles $\{U_1, U_2, ..., U_n\}$, where U_i contains m features, and each feature belongs to a feature group $fg \in \{fg_1, fg_2, ..., fg_s\}$, and $d \in \mathbb{R}$

for $i = 0 \rightarrow n$ **do**
  Compute the mean μ_{fg_i} and standard deviation σ_{fg_i} for all feature groups $fg_i \in \{fg_1, fg_2, ..., fg_k\}$.
  for $j = 0 \rightarrow m$ **do**
    for $k = 0 \rightarrow s$ **do**
      $lb \leftarrow \mu_{fg_k} - (d \cdot \sigma_{fg_k})$
      $ub \leftarrow \mu_{fg_k} + (d \cdot \sigma_{fg_k})$
      if $U_i[j].fg == fg_k$ AND ($U_i[j] < lb$ OR $U_i[j] > ub$) **then**
        $U_i[j] = 0$
return $\{U_1, U_2, ..., U_n\}$

the true mean and median values are selectively chosen to represent single feature values. If the number of samples for a given feature f is less than k, then the true mean is used. Otherwise, the true median from the set of collected values is used. We discuss the improvement of this effect in Section IV. An example procedure for the derivation of the average key press time is shown in Algorithm 2.

Algorithm 2 Stabilized feature value selection

Require: A set of feature samples $F = [f_1, f_2, ..., f_n]$ and a boundary $k \in \mathbb{N}$

$val \leftarrow 0$
if $n < k$ **then**
  $val \leftarrow$ the average of all features in $F = [f_1, f_2, ..., f_n]$
else
  if $n \mod 2 == 0$ **then**
    $val \leftarrow (F[(n/2) - 1] + F[n/2])/2$
  else
    $val \leftarrow F[n/2]$
return val

Finally, the last data filtering technique we propose is the removal of similar feature measures between pairs of user profiles generated during the training period. After the user profiles from the training period are generated, we run Algorithm 3. This procedure walks the columns of user profiles, computes their mean μ and standard deviation σ, and then reduces all column elements that fall within the range $[\mu - d\sigma, \mu + d\sigma]$, where $d \in \mathbb{R}$. Assuming a normal distribution of features across all user profiles, this will effectively leave only the outlier elements in each profile, thus increasing the uniqueness of each user profile (shown in Figure 9).

C. Feature Weight Derivation

The derivation of feature weights is a topic that is seldom discussed in the literature related to this problem. It appears as though weight function $w : \mathcal{F} \rightarrow \mathbb{R}$ is defined by intuition. Tools such as WEKA can help guide the selection of weights by performing attribute selection on the set of features to determine those which yield maximum information gain. With this tool, weights can be assigned in decreasing order proportional to the amount of information gain. For example, if the attribute selection process reveals that m features out of a set of n are most important, we can assign weights according to the following function:

$$w(f, S) = \begin{cases} f \cdot S[f] & : f \in S \\ f & : f \notin S \end{cases}$$

Algorithm 3 Pairwise similarity pruning

Require: A set of user profiles $\{U_1, U_2, ..., U_n\}$, where U_i contains m features, and $d \in \mathbb{R}$

```
for i = 0 → m do
    Compute the mean μ and standard deviation σ for feature f_i across all profiles U_j, 1 ≤ j ≤ n
    for j = 0 → n do
        lb ← μ − (d · σ)
        ub ← μ + (d · σ)
        if U_j[i] > lb AND U_j[i] < ub then
            U_j[i] = 0
return {U_1, U_2, ..., U_n}
```

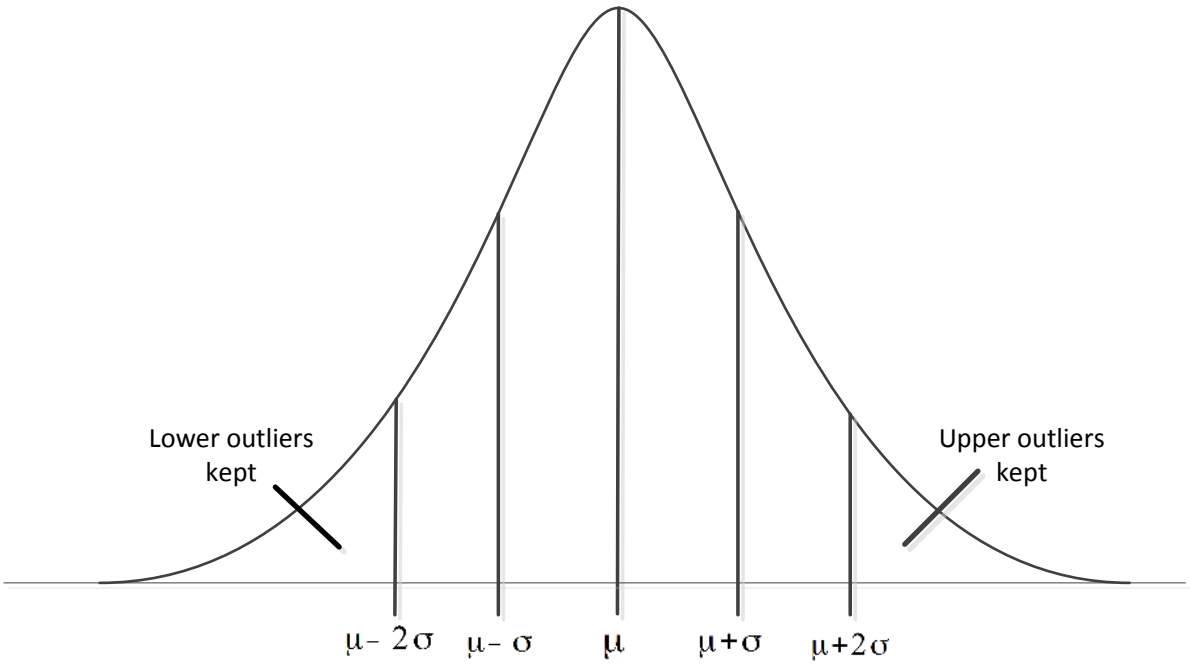

Fig. 9. The outliers selected from the pairwise similarity pruning algorithm where $d = 2$.

In these equations, $f \in \mathcal{F}$ and S is the set of selected attributes sorted in increasing order of priority. Also, $S[f]$ returns the index of f in S. Again, due to time constraints, we were not able to explore this topic in much detail, but is left open for future work.

D. Classification

In the evaluation system only the standard Euclidean distance classification technique was implemented due to its simplicity with prototyping and testing new feature sets. Other classification experiments were conducted with the use of WEKA. Of the many supervised learning algorithms provided by this tool, we chose to make use of the ZeroR, BayesNet, J-48, LADTree, BTree, Random Forest, and NBTree classification algorithms to determine which yielded the highest accuracy with a high degree of confidence. The results from these experiments are presented in Section IV.

IV. Experimental Analysis

In this section we present the results from our experiments on the evaluation system, feature extraction, and data filtering techniques produced during the project. These results show how we evolved the evaluation system from a static, fixed-text identification system to one that is based on dynamic, free-form input. We provide all rationale for system design adjustments, including feature extraction and data preprocessing changes, alongside our experimental results.

A. System Evolutions

Over the course of this project, we experimented with several feature extraction and data filtering techniques. We describe the main features and experiments conducted with each evolution in the following sections. The first three feature set evolutions were analyzed in the context of our evaluation system, but due to the difficulty of conducting extensive tests and the lack of available test subjects, we switched to WEKA for the final evolution of the feature set.

1) Feature Set 0: The first feature set consisted of the following elements:

- Single key press time (minimum, maximum, average, standard deviation)
- Single key fly time (minimum, maximum, average, standard deviation)
- Digraph time (minimum, maximum, average, standard deviation)

No data preprocessing or filtering techniques were employed because we wanted to gauge the standalone value added by each of these features. The experiments we conducted with this feature set were done in the context of the evaluation system, which made use of a trivial classifier based on the Euclidean distance between pairs of user profiles (U_i, U_j), where $U_i, U_j \in \mathcal{U}$ (the set of user profiles that were obtained through the training process). The identified user U_{id} was determined by $U_{id} = \min_{U_i, U_j \in \mathcal{U}} d(U_i, U_j)$, where d is the Euclidean distance function defined for two vectors.

Due to the expected poor performance of this feature set, we only conducted small experiment where at most 10 typing samples were extracted from the user during their session and used to check their identity. We present the results from one experiment in Table II. This data shows an identification accuracy rate of approximately 10%, which confirmed our original hypothesis that more sophisticated feature extraction and data filtering techniques are needed to remove outliers and statistical anomalies embedded in the user profiles.

TABLE II
IDENTIFICATION RESULTS FROM THE EXPERIMENT WITH *Feature Set 0*. ALL ENTRIES IDENTIFIED AS *Chris* ARE CORRECT.

Experiment	Identified User
1	Douglas
2	Douglas
3	Douglas
4	Douglas
5	Douglas
6	Douglas
7	Douglas
8	Chris
9	Douglas
10	Douglas

2) Feature Set 1: In response to the poor performance of the first feature set and based on our physical observations of users going through the training process with our system, we chose to add two additional features that would provide more information within user profiles: relative typing speed and error rate. Collectively, this feature set was composed of the following elements:

- Single key press time (minimum, maximum, average, standard deviation)
- Single key fly time (minimum, maximum, average, standard deviation)
- Digraph time (minimum, maximum, average, standard deviation)

- Relative typing speed
- Total number of errors (deletions)

We also introduced some elementary data filtering techniques to help improve our feature extractor and standard Euclidean classifier. The pruning techniques included removing zero-valued features from profiles and normalizing (digitizing) all entries to the $[0, 1]$ range before performing classification. We then conducted the same experiment as with the previous feature set but extended the number of samples to 15. Our results are shown in Table III, and indicate an improvement in the identification accuracy from 10% to approximately 33%. This was an indication that normalization and elementary outlier removal are helpful when trying to improve the classification results.

TABLE III
IDENTIFICATION RESULTS FROM THE EXPERIMENT WITH *Feature Set 1*. ALL ENTRIES IDENTIFIED AS *Chris* ARE CORRECT.

Experiment	Identified User
1	Chris
2	Chris
3	Chris
4	tjf
5	tjf
6	tjf
7	Chris
8	tjf
9	tjf
10	tjf
11	tjf
12	Chris/Sarah
13	tjf
14	tjf
15	tjf

3) Feature Set 2: For this evolution of the system we trimmed the dimensionality of the feature sets contained within user profiles by removing the minimum and maximum values for each single key press time, fly time, and digraph time. This made our classification techniques more heavily weighted on the average and standard deviation calculations for a given feature, which seemed to be much more stable. Furthermore, we added the feature stabilization technique that chooses between the average or median value of set of feature samples depending on the number of samples that were collected. For instance, if our sample cutoff was $k = 5$, then for all feature sample sets that contained less than 5 elements we would use the average of the feature sample set in the user profile. Conversely, if the feature sample set contained at least 5 elements, then we would select the true median of this set to store in the user profile. Statistically speaking, the median would tend to be more stable with larger feature sample sets, thus reducing the possibility of statistical anomalies affecting our results.

Additionally, we also added some logic in our feature extractor to remove digraph data that would normally be collected on sentence boundaries. Since typical users tend to take intermittent breaks at the end of sentence the digraph time for keystrokes that cross this boundary could increase significantly if not removed.

Collectively, this evolution of our feature set was composed of the following elements.

- Single key press time (average/median, standard deviation)
- Single key fly time (average/median, standard deviation)
- Digraph time (average/median, standard deviation)
- Relative typing speed
- Total number of errors (deletions)

Again, we conducted the same experiment on a set of 15 classification samples. A subset of the results are shown in Table IV. At this point, we had an identification accuracy of approximately 40%, which was good enough for us to start using WEKA to streamline the experimentation process for the remainder of the project.

TABLE IV
IDENTIFICATION RESULTS FROM THE EXPERIMENT WITH *Feature Set 2*. ALL ENTRIES IDENTIFIED AS *Chris* ARE CORRECT.

Experiment	Identified User
1	Chris
2	Chris
3	Chris
4	tjf
5	tjf
6	tjf
7	Chris
8	tjf
9	tjf
10	Chris
11	tjf
12	Chris/Sarah
13	tjf
14	tjf
15	tjf

4) Feature Set 3: For this feature set evolution we began using WEKA as the primary analysis tool for our feature extraction and data filtering techniques. As previously mentioned, we chose to experiment with the ZeroR, BayesNet, J-48, LADTree, BTree, Random Forest, and NBTree classification algorithms (as opposed to simple Euclidean distance classifier). It is important to note that the same set of data was used for both training and testing (due to a lack of data acquired from users). One of the most significant improvements for this version of the feature set was the addition of the aforementioned feature group statistical outlier removal algorithm. Without changing the elements in the feature set, we tested the addition of this filtering technique using WEKA. The results from these classification experiments are shown by the blue bars in Figure 10.

In an attempt to improve the results we tried to rely on the WEKA attribute selection features as a means of generating feature weights (that is, we trimmed the feature set dimensionality based on the attributes recommended by WEKA, thus simulating the assignment of zero-weight values to the omitted features). Using WEKA's attribute selection tool we identified 53 (out of 537) features that yielded maximum information gain for the profiles generated from *Feature set 3* using the $CfsSubsetEval$ algorithm [9]. This algorithm evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. It was interesting to find that single key duration features were weighted higher than digraph time features.

Further experimentation revealed that user profile reduction based on the attribute selection results did not change classification accuracy, with the exception of the J-48 algorithm, which actually decreased from 46.2% to 38.5%. This change is shown by the red bar in Figure 10.

Various modifications were made to the classification algorithms themselves as well, with the most successful adjustment being an increase in the minimum number of objects in leaf nodes and pruning settings for tree-based algorithms. The results from these changes are shown by the green bars in Figure 10. If our data set for training and testing was larger, this tree expansion might have lead to significantly worse performance. Future work will attempt to verify this hypothesis. Our results indicate that pairwise pruning among all profiles actually decreased the classification accuracy.

5) Feature Set 3 with Explicit Test Data: After obtaining more user data to derive explicit training and test sets we wanted to repeat the previous experiment. However, this time we focused on the utility gained from the aforementioned pairwise similarity pruning algorithm. To do this, we first removed this technique from the data filtering step and then conducted the WEKA classification experiment. In this experiment, only the new feature group statistical outlier removal algorithm was applied to filter the data. Our results are shown in Figure 11. Then, we repeated the experiment with both the feature group statistical outlier and pairwise similarity filtering algorithms. The results from this experiment are shown in Figure 12.

It is interesting to note the difference between the J-48 decisional tree that is constructed based on the

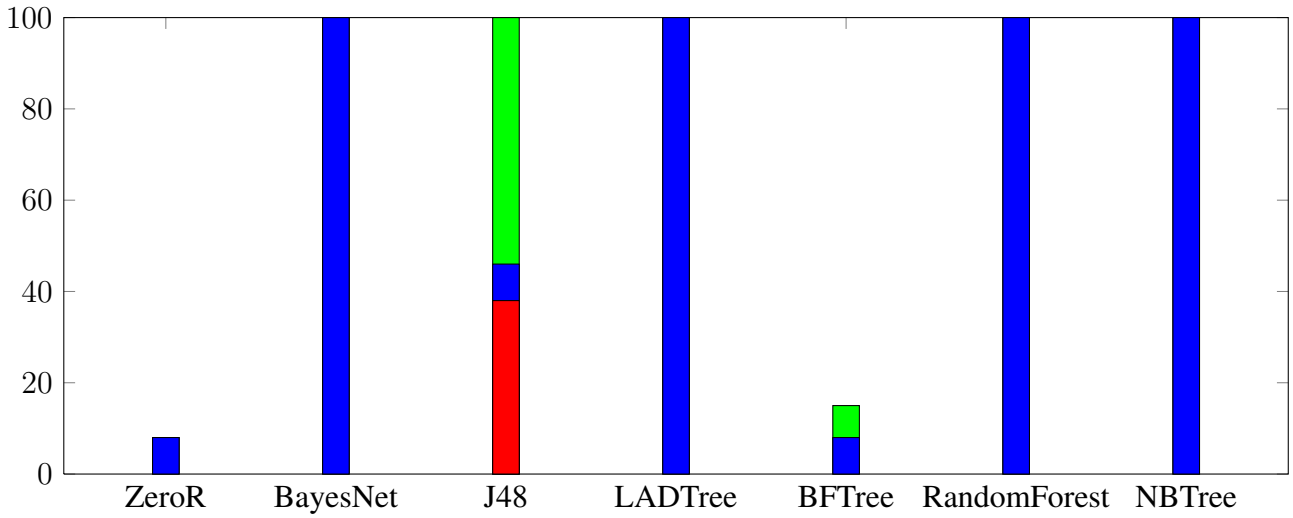

Fig. 10. Classification results for the *Feature Set 2* and *Feature Set 3*. The blue bars indicate the accuracy of classification techniques using *Feature Set 2*, the red bars indicate the difference introduced from *Feature Set 2* to *Feature Set 3*, and the green bars indicate the improvements gained by adjusting the parameters of the respective classification algorithms. The high accuracy percentages for this data set is due to the fact that the testing data was also the training data.

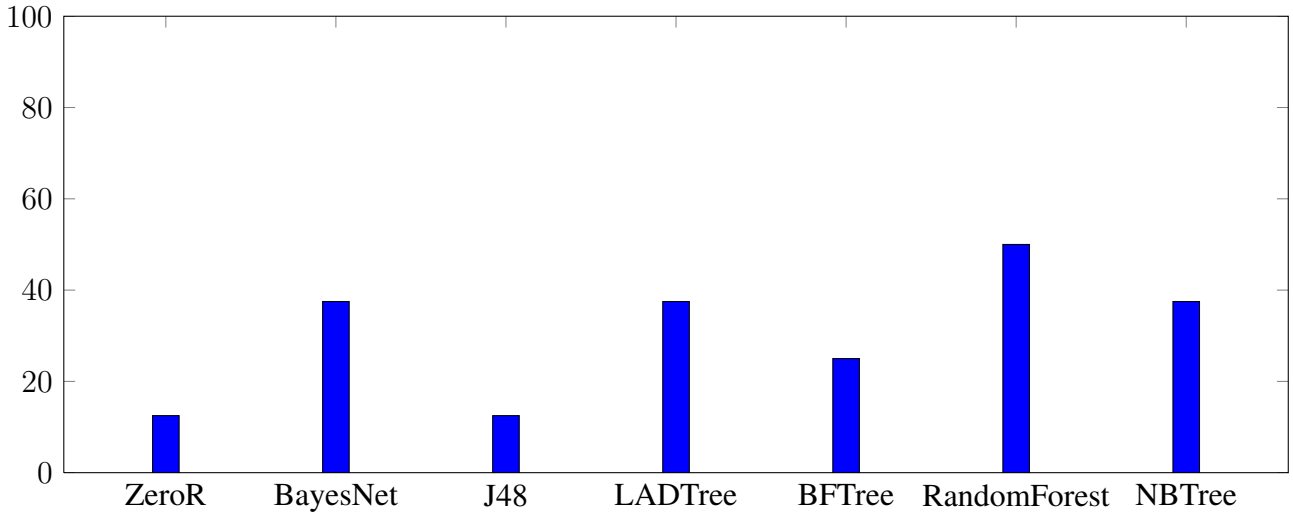

Fig. 11. WEKA-based classification results using information collected using *Feature Set 3* without the help of the pairwise similarity pruning technique.

pairwise similar filtered and unfiltered data. Using WEKA, we were able to extract these trees in order to determine how our data filtering techniques utilize the most significant attributes (in terms of information gain). The resulting trees are shown in Figures 13 and 14. From these models we were able to conclude the following:

- The typing speed feature was the largest identifying attribute belonging to both the filtered and unfiltered feature sets. This came as a surprise to us because WEKA's attribute selection algorithms tend to place this as a less important feature behind single key press duration averages and standard deviations.
- The trees are not equally balanced. The pre-filtering tree in Figure 13 has a much deeper hierarchy on the left part of the tree than on the right. However, the post-filtering tree in Figure 14 is, by definition, balanced. This seems to indicate that our filtering techniques were effective in making each profile more unique because a balanced model implies greater differentiation between user profiles.

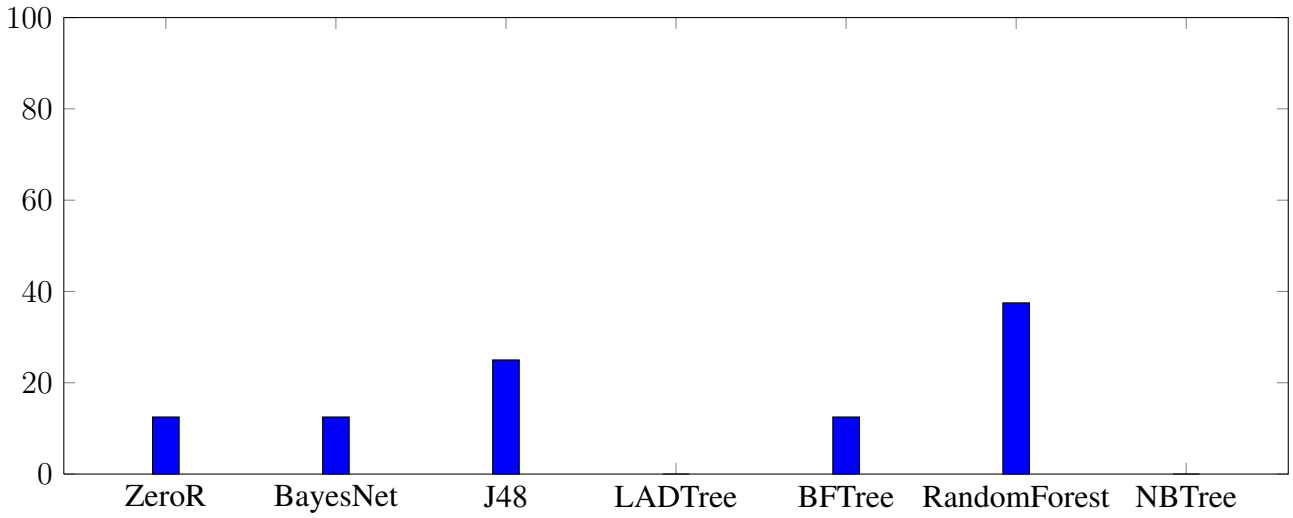

Fig. 12. WEKA-based classification results using information collected using *Feature Set 3* with the help of the pairwise similarity pruning technique. All classification methods decreased in performance, with the exception of the J-48 algorithm.

- The pre-filtering tree has less leaves than the post-filtering tree. Furthermore, neither of the trees has a leaf for every user profile in the training set. This immediately implies that the post-filtering model will have a better classification accuracy than the pre-filtering model, and also that neither model can possibly yield an identification accuracy of 100%.

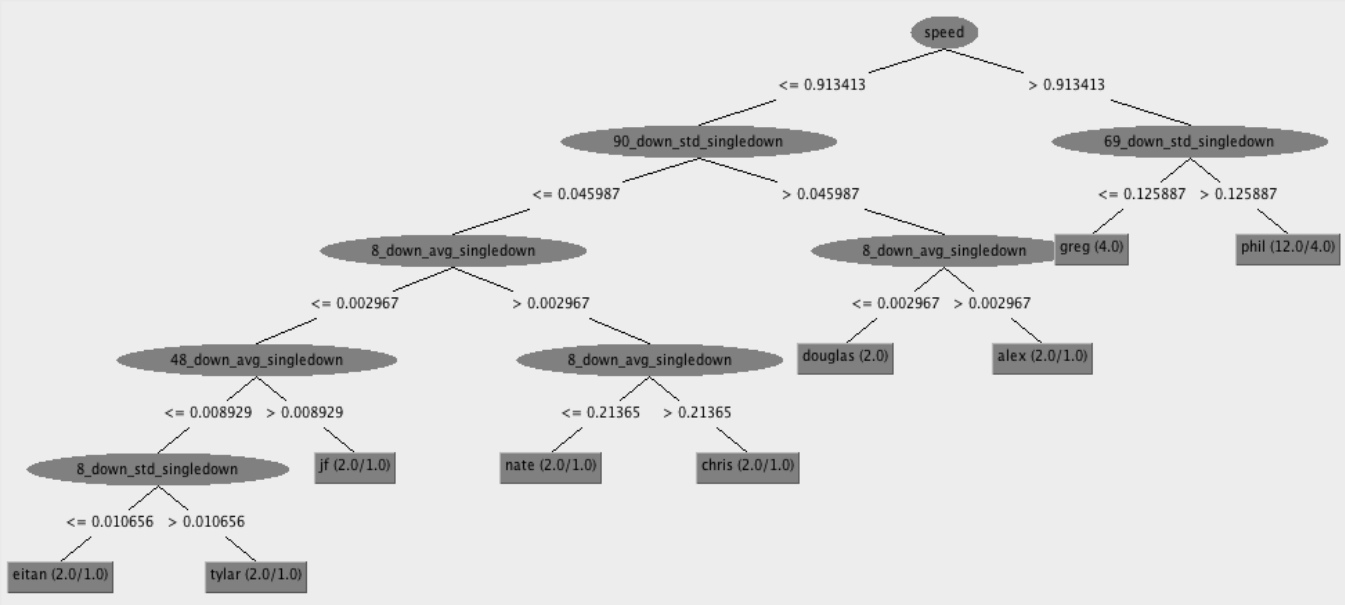

Fig. 13. The J-48 decision tree built using training data from non-filtered profiles composed of features from *Feature Set 3*.

B. Context-Sensitive User Profiles

One of the biggest obstacles to accurate identification systems based on free-form text input is the psychological mindset for the host user when interacting with the system. To our knowledge, there has not been research conducted on this topic in the literature, so we present a novel thread of research. In order to study the variance of user typing patterns in different scenarios (which we refer to as contexts),

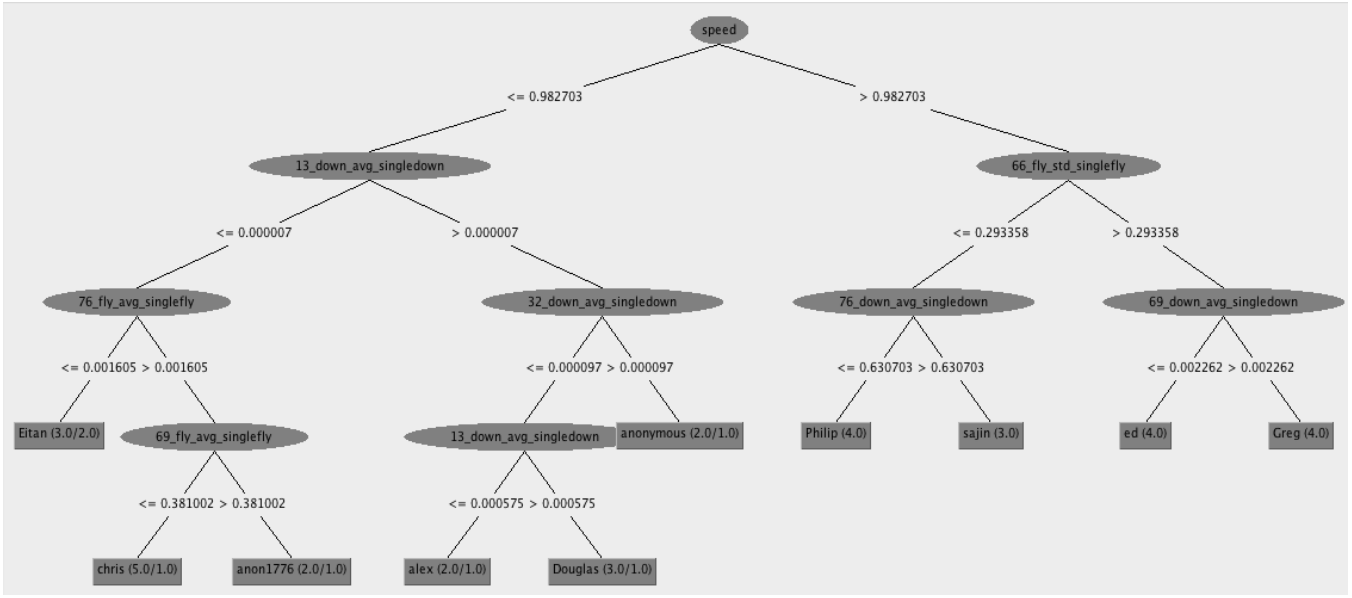

Fig. 14. The J-48 decision tree built using training data from filtered profiles composed of features from *Feature Set 3*.

we asked a sample of 7 computer science and software engineering students to record their keystroke information when performing the following tasks:

1) Transcribe a fixed set of text. In this case, it was a text snippet taken from the novel "Huckleberry Finn".
2) Transcribe content that is presented in a multimedia format (e.g. a movie or television show).
3) Write freely about a subject of their own choosing for $4 - 5$ minutes.
4) Write three programs that accomplish the following tasks:
   - Print "Hello, World!"
   - Compute the nth Fibonacci number
   - Solve the "FizzBuzz" problem

As expected, our experiments indicated that users exhibit different typing patterns depending on the context in which they are using the system. This was supported by the distance between user context profiles generated from each of these tests. The first implication of this result is that user profiles must be context sensitive, either by assigning weights to the features most relevant for a given context or by filtering the non-relevant features from the profile. The latter approach would theoretically remove redundant data from the profiles and thus improve the training results. We further experimented with this idea by employing our feature group outlier and pairwise similarity pruning algorithms to context profiles collected for the 7 users. It was our goal to see if these filtering techniques helped make different user context profiles more unique. Our results from this experiment are shown in Table V.

TABLE V
A DEPICTION OF THE DATA COLLECTED FROM OUR CONTEXT-SENSITIVE EXPERIMENTS. EACH VALUE REPRESENTS AN AVERAGE EUCLIDEAN DISTANCE BETWEEN PAIR OF USER PROFILES. EACH DIFFERENCE WAS CALCULATED USING ONLY PROFILES FROM THE SAME USER.

Task ID Pair	Nonpruned average	Pruned average	Average Difference
1,2	1251.018	885.148	365.870
1,3	1182.874	795.290	387.583
1,4	1171.213	791.771	379.442
2,3	1195.895	813.317	382.578
2,4	1237.885	947.573	290.312
3,4	1340.510	1192.287	148.222

As one can see, the application of these filtering techniques on average reduced the distance between context profiles for a single user. The immediate implication from this result is that non-relevant features were removed from the different context profiles, and the only features that contributed to the distance between a pair of profiles were those that were relevant to each individual context (i.e. key press duration for '' and '' symbols for a programming context profile and common puncuation marks for the transcription context profiles). In order to determine if these results held for other distance measurements, we conducted experiments with the City Block, Chebyshev, and Cosine distance metrics, defined below.

$$CityBlockDistance(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

$$CosineDistance(x, y) = 1 - \frac{x \cdot y}{\sqrt{(x \cdot x)(y \cdot y)}}$$

$$ChebyshevDistance(x, y) = \max_i\{|x_i - y_i|\}$$

As shown in Tables VI, VII, and VIII, the pruning techniques yielded drastic reductions in context differences for users. After filtering the datasets generated from *Feature Set 3* using the pairwise similarity and feature group outlier removal algorithms, the different user context profiles were all closer to each other (on average). This implies that filtering is a vital step in dynamic free-form text input systems where user goals may vary over time depending on the context in which they use the system.

TABLE VI
A DEPICTION OF THE DATA COLLECTED FROM OUR CONTEXT-SENSITIVE EXPERIMENTS. EACH VALUE REPRESENTS AN AVERAGE CITY BLOCK DISTANCE BETWEEN PAIR OF USER PROFILES. EACH DIFFERENCE WAS CALCULATED USING ONLY PROFILES FROM THE SAME USER.

Task ID Pair	Nonpruned average	Pruned average	Average Difference
1,2	286442461.490	5514.290	286436947.199
1,3	286774657.250	4379.517	286770277.732
1,4	287035176.622	5418.453	287029758.168
2,3	413615.546	4662.060	408953.486
2,4	669930.652	4705.601	665225.051
3,4	413836.198	78424.067	335412.131

TABLE VII
A DEPICTION OF THE DATA COLLECTED FROM OUR CONTEXT-SENSITIVE EXPERIMENTS. EACH VALUE REPRESENTS AN AVERAGE COSINE DISTANCE BETWEEN PAIR OF USER PROFILES. EACH DIFFERENCE WAS CALCULATED USING ONLY PROFILES FROM THE SAME USER. EACH USER PROFILE IS COMPOSED OF TIME MEASURES FOR THE FEATURES DESCRIBED IN SECTION III

Task ID Pair	Nonpruned average	Pruned average	Average Difference
1,2	0.555	0.353	0.201
1,3	0.459	0.334	0.125
1,4	0.688	0.390	0.297
2,3	0.488	0.328	0.160
2,4	0.512	0.275	0.236
3,4	0.774	0.477	0.297

To test the effectiveness of these pruning techniques on groups of context profiles from the same user before being used in the training process, we performed the following experiment:

1) Filter the sets of context profiles for each user who completed the four training tasks.
2) Group all of these filtered profiles together and then run the pruning algorithms on this larger group.
3) Perform the previously discussed WEKA classification experiment with this new training set.

The results from this experiment were very promising. We saw in increase in the overall classification accuracy for all classification algorithms. Our results are shown in Figure 15, where we also tried to vary the value of d in the pairwise similarity pruning algorithm to quanitfy its impact on the classification results.

TABLE VIII
A DEPICTION OF THE DATA COLLECTED FROM OUR CONTEXT-SENSITIVE EXPERIMENTS. EACH VALUE REPRESENTS AN AVERAGE CHEBYSHEV DISTANCE BETWEEN PAIR OF USER PROFILES. EACH DIFFERENCE WAS CALCULATED USING ONLY PROFILES FROM THE SAME USER. EACH USER PROFILE IS COMPOSED OF TIME MEASURES FOR THE FEATURES DESCRIBED IN SECTION III

Task ID Pair	Nonpruned average	Pruned average	Average Difference
1,2	286395221.970	533.0291	286395221.971
1,3	286762725	572.808	286762152.192
1,4	287004837.5	768.826	287004068.674
2,3	366970	599.518	366370.482
2,4	609082.5	539.736	608542.764
3,4	383408	71244.658	312163.342

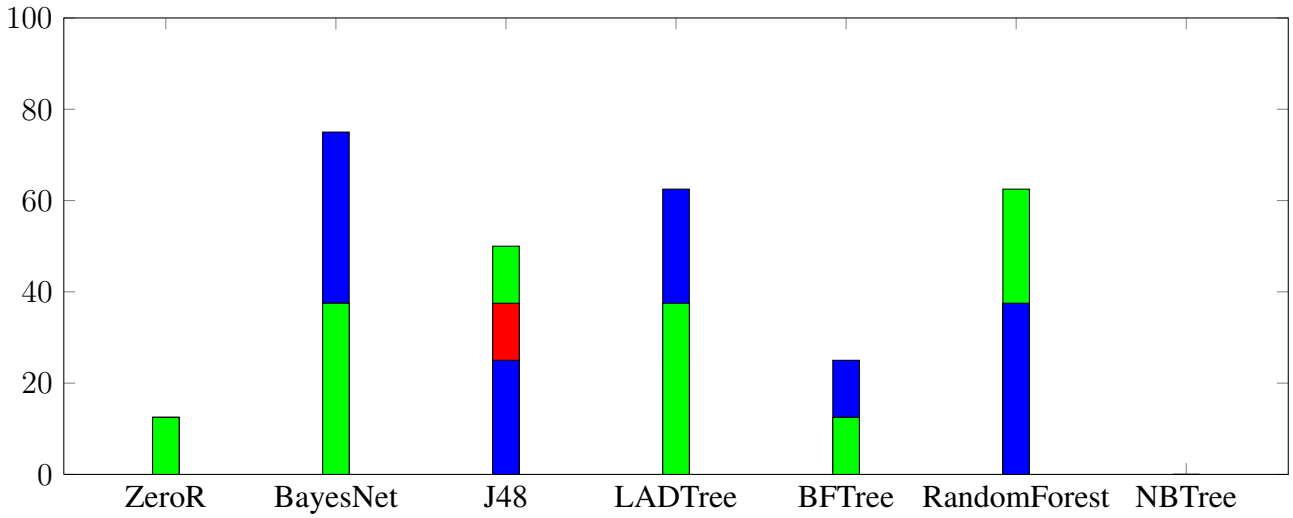

Fig. 15. WEKA-based classification results using information collected using *Feature Set 3* with the help of the feature group outlier and pairwise similarity pruning techniques on user context profiles before the training process. The BayesNet and NBTree classification algorithms were unable to produce any results in this experiment.

V. TECHNIQUE POSTMORTEM

Based on our experiments, we can make many suggestions for future research directed towards improving the identification accuracy (and efficiency) of keyboard-based continuous authentication systems that use dynamic free-form text input. Perhaps the biggest takeaway from this project is that the statistical stability of the feature set is critical to achieving good results. Our first two feature set definitions contained the minimum and maximum value for every single key press time, single key fly time, and digraph time. After these were removed the classification results saw an improvement from 33% (Euclidean classifier) to anywhere from 40% (Euclidean, *Feature Set 2*) to $60-80\%$ (WEKA classifiers, *Feature Set 3*) with the third and fourth iteration of the feature sets (which omitted the minimum and maximum values). Thus, we can safely conclude that the high degree of variance in the minimum and maximum value for a given user over a period of time was detrimental to the identification accuracy.

We can also make several claims about the effectiveness of various feature extraction and user profile filtering techniques. For brevity, we list them below:

- Statistical pruning by feature groups helped bring user profiles closer to the true "normal" behavior exhibited by users. Although it may be argued that this technique removed important information about key press durations for infrequently used keys, we felt it was an important technique to use for dynamic free-form text input systems because the user is not expected to consistently maintain their "normal" typing patterns during the data collection process. Without this pruning technique, the change in user behavior and typing patterns might make their testing profile significantly different from the training profile, and we want to avoid that case.

- Feature group outlier and pairwise similarity pruning among user profiles was effective when applied to different context profiles from the same user. This was motivated by the goal of reducing the effect of typing pattern changes based on specific use cases. As discussed in the previous section, this helped improve the identification accuracy of all classifiers tested under WEKA (with the exception of the LADTree).
- Generally speaking, feature set reduction based on attribute selection did not seem to improve the identification results (as shown by the experiment conducted with only WEKA-identified features in user profiles). This was a surprising result, as it was expected that a reduction in the dimensionality of the feature set for a profile would lead to smaller J-48 decisional trees with more information gain in the non-leaf nodes. However, the reduction in identification accuracy from 46.2% to 38.5% clearly refutes this hypothesis.
- Larger training sets with more diverse user context profiles had a drastic impact on the classification accuracy of a variety of algorithms. Our original experiments were limited in scope due to the lack of available data, but as more external users were enrolled in the system, our experiments began to show true differences in the classification accuracy with the large sample size experiments.

We are unable to make definite claims about the effectiveness about normalization on the feature sets, as we tried this technique in conjunction with the elementary filtering of removing zero entries from user profiles. However, since normalization is a common unsupervised filtering technique used in many applications, we hypothesize that it certainly helped improve our results (rather than hurt them).

Since the best results from this project came from the experiments conducted with context-sensitive user profiles, we conclude that the training process for a continuous authentication system should require users to cover a variety of use case scenarios (rather than just their normal typing behavior). Pairwise similarity pruning and feature group statistical outlier removal on the generated profiles from these different contexts can then be used to construct a single user profile that captures important information about the user's typing patterns in a variety of contexts.

Overall, we observed that the following combination yielded best identification accuracy over the course of this project.

- Using the features described in *Feature Set 3*.
- Using sentence boundary awareness, stabilized feature value selection, feature group outlier removal, and pairwise similarity pruning to improve the quality of user profiles.
- Pruning user context profiles before the training process that is performed across all user profiles.
- Using the LADTree or RandomForrest classification algorithms to perform the identity checks.

In order to improve the confidence and accuracy of our identification results we would need to increase the number of test subjects from which we gather data for our experiments. The variability in our results across different feature sets was likely influenced by the small number of people who participated in the experiments.

VI. Conclusion

In this paper we have presented our evaluation platform for a dynamic, free-form text input identification system based on keyboard biometrics. We discussed our novel feature extraction techniques and describe why they differ from past approaches, and then we provide an in-depth discussion of the experiments conducted to test our system. The WEKA machine learning and data mining tool was vital to such experiments because it enabled us to rapidly test new feature set evolutions and feature extraction techniques offline.

We also presented the idea of context-sensitive user profiles. Based on our intuition and the experiments we conducted, user typing patterns are extremely sensitive to the context in which the user is interacting with the system. We have shown that feature weights can (and should) be dynamically generated based on these contexts. Furthermore, we have presented several data filtering techniques that can be used to

bring user profiles from different contexts closer together with the hopes of improving the classification accuracy when compared against other user profiles.

The biggest shortcoming in this work is the size of the samples used as the basis for our experiments. Our future work will consist of expanding our experimental database to include more user profiles, and then re-running our experiments with this new collection of data. It is our expectation that this will remove any statistical anomalies that may be present in our current results.

VII. Acknowledgements

The author would like to thank Dr. Leonid Reznik of the Department of Computer Science at RIT for providing helpful guidance and assistance during this project. In addition, the author would like to thank Nate Smith and Eitan Romanoff for providing initial source code and material that motivated the exploration into identification systems based on dynamic free-form text input.

References

[1] S. Banerjee and D. Woodard, "Biometric authentication and identification using keystroke dynamics: A survey," *Journal of Pattern Recognition Research*, vol. 7, pp. 116–139, 2012.

[2] A. Jain, A. Ross, and S. Prabhakar, "An introduction to biometric recognition," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 14, no. 1, pp. 4 – 20, jan. 2004.

[3] J. Marsters, "Keystroke dynamics as a biometric," Ph.D. dissertation, University of Southampton, 2009.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[5] R. Spillane, "Keyboard apparatus for personal identification," *IBM Technical Disclosure Bulletin*, vol. 17, 1975.

[6] F. Monrose and A. Rubin, "Authentication via keystroke dynamics," in *Proceedings of the 4th ACM conference on Computer and communications security*, ser. CCS '97. New York, NY, USA: ACM, 1997, pp. 48–56. [Online]. Available: http://doi.acm.org/10.1145/266420.266434

[7] P. Dowland, S. Furnell, and M. Papadaki, "Keystroke analysis as a method of advanced user authentication and response," in *Proceedings of the IFIP TC11 17th International Conference on Information Security: Visions and Perspectives*, ser. SEC '02. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2002, pp. 215–226. [Online]. Available: http://dl.acm.org/citation.cfm?id=647185.719834

[8] "Computer user verification using login string keystroke dynamics," *IEEE Transactions on Systems, Man, and Cybernetrics - Part A: Systems and Humans*, vol. 28, pp. 236–241, 1998.

[9] M. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, The University of Waikato, 1999.