

Developing a Java-Based Keystroke Biometric System for Long-Text Input

Giang Ngo, Justin Simone, Huguens St. Fort
Computer Science Department, Pace University
1 Martine Avenue, White Plains, NY 10606, USA
{gn90004n, js97978w, hs78586w}@pace.edu

Abstract

Java-based feature extraction and pattern classification programs were developed for a Pace University CSIS Doctorate of Professional Studies student completing her thesis on long-text-input keystroke biometrics. Although the general functionality of these programs was developed for a prior DPS researcher using the SAS programming language, the current Java-based system was developed for added functionality, increased usability, and ease of learning. To optimize the results in ideal and application-oriented conditions the programming team worked with the researcher to increase the feature set, optimize the pattern classifier, and provide options for different fallback scenarios for small-samples. The current feature extraction program has 239 features and five parameters to optimize, and the pattern classifier has two procedural modes with seven parameters to enable/disable various feature sets.

1. Introduction

The goal of this paper is to explain the methodology of developing and optimizing keystroke biometric and pattern classification Java programs building upon, as the baseline for correctness, the results of an earlier SAS program that calculates simplified results.

These programs, in conjunction with a Java applet for data capture, can be used by any researcher attempting to identify keystroke patterns in long-text passages. Potential users could include professors of online courses who need to validate the work submitted by students. A paper by Villani et al. [3] explores the results when a subject trains on one style of keyboard (e.g., desktop keyboard, laptop keyboard) and/or one style of data entry (e.g., free text, copy task) and is tested against the same and/or a different style of keyboard and/or data entry style.

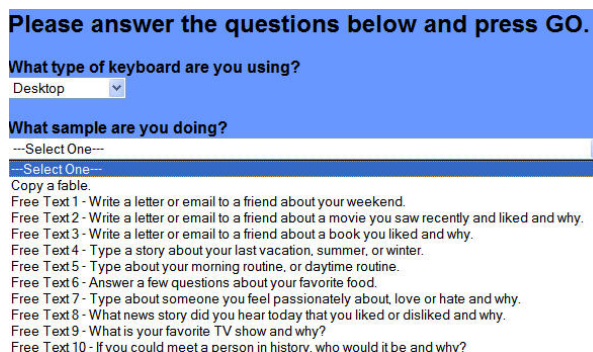
2. Keystroke Biometric System

The keystroke biometric system consists of three components: the Java applet for collection of raw data; the feature extractor; and the pattern classifier.

2.1 The Java Applet and Raw Data Collection

To begin the data collection process the user accesses a Web site hosted by a server with the capability of serving HTML and PHP files, and running a MySQL database. The user is initially required to register with the Web site and is entered in the MySQL database. To allow users to leave and return to the Web site, four counters are also initialized to track the entry number the user will begin with upon returning to the Web site. The four experimental categories are copy task on a desktop, copy task on a laptop, free-text entry on a desktop and free-text entry on a laptop.

At the completion of registration or, upon returning to the site, the user is redirected to the activity selection PHP page. This page receives the user's first name and last name from the referring page and queries the database to obtain the values of the counter fields.



Please answer the questions below and press GO.

What type of keyboard are you using?
Desktop

What sample are you doing?
--Select One--
Copy a fable.

Free Text 1 - Write a letter or email to a friend about your weekend.
Free Text 2 - Write a letter or email to a friend about a movie you saw recently and liked and why.
Free Text 3 - Write a letter or email to a friend about a book you liked and why.
Free Text 4 - Type a story about your last vacation, summer, or winter.
Free Text 5 - Type about your morning routine, or daytime routine.
Free Text 6 - Answer a few questions about your favorite food.
Free Text 7 - Type about someone you feel passionately about, love or hate and why.
Free Text 8 - What news story did you hear today that you liked or disliked and why.
Free Text 9 - What is your favorite TV show and why?
Free Text 10 - If you could meet a person in history, who would it be and why?

Figure 1: Activity Selection page.

The user is required to select the style of keyboard being used and whether he/she will be completing a free text or copy text sample as indicated in the drop-down box (Figure 1). Clicking go redirects the user to the appropriate Java applet based on his/her selections (Figure 2). There are six

pieces of information sent to, and required by, the Java applet: first name; last name; experiment style (e.g., free text, copy task); sequence number for the selected experiment style (respective counter field value); keyboard style; and awareness. Awareness refers to whether the user knows he/she is working with a keystroke biometric system. If the Java applet does not receive these six values, or if the user does not have a Java Runtime Environment (JRE) equal to or later than version 1.4, the applet will not launch. Lastly, the user must use Microsoft's Internet Explorer in order for the applet to function properly.

After analyzing previous raw data files, it was identified that typos or inconsistencies in a participant's name causes problems in the feature extractor. By requiring the user to register once and use the same first and last name to access the system, the problem is eliminated. The same principle is true for the activity sequence number; should the user enter a number already used, the user will overwrite his/her existing raw data file. This is corrected through the use of counters in the database managed through PHP scripts.

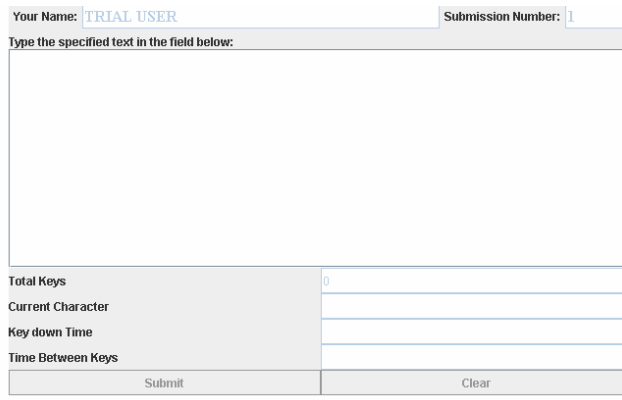


Figure 2: Java applet before any keystrokes have been entered.

Depending on the sample being collected, the system checks for a minimum number of keystrokes. In the study by Villani et al. [3] the copy task entries must be at least 635 keystrokes and free text samples at least 677 keystrokes, otherwise the user is prompted to continue typing (Figure 3).

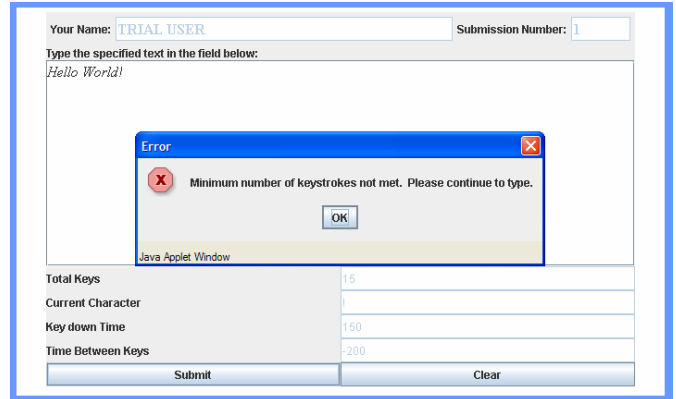


Figure 3: Warning if user clicks *submit* before meeting the minimum number of keystrokes.

When the user correctly completes the task and clicks *submit*, a PHP file is called, which writes the raw data information to a text file (Figure 4) and (transparent to the user) updates the user's counter field by one in the database. The user sees the Java applet in a nearly identical state as that pictured in Figure 2, except the sequence number has been incremented. The user can enter another sample or click the back button to return to the activity selection page.

For ease of locating the raw data files, each experimental style/keyboard combination is given its own directory on the server. Before progressing to the feature extraction process, the researcher must FTP the raw data files to a directory on his/her local disk.

NewUser	Submission 1	Entry #	Key	Keycode	Location	Press	Release
		Num 1	?	Shift	2	1114450735680	1114450736962
		Num 2	H	H	1	1114450735991	1114450736311
		Num 3	e	E	1	1114450737653	1114450738144
		Num 4	l	L	1	1114450738735	1114450739256
		Num 5	l	L	1	1114450739786	1114450740277
		Num 6	o	O	1	1114450740998	1114450741399
		Num 7		Space	1	1114450742090	1114450742420
		Num 8	?	Shift	2	1114450743542	1114450745004
		Num 9	w	W	1	1114450743872	1114450744263
		Num 10	o	O	1	1114450745755	1114450746216
		Num 11	r	R	1	1114450747017	1114450747437
		Num 12	l	L	1	1114450748138	1114450748549
		Num 13	d	D	1	1114450749310	1114450749771
		Num 14	?	Shift	2	1114450751373	1114450753776
		Num 15	!	!	1	1114450752445	1114450752885
		Left Clicks	0				
		Right Clicks	0				
		Double Clicks	0				

Figure 4: Aligned raw data file for "Hello World!" [1]

2.2 Feature Extraction

The software developers use Borland's JBuilder as the IDE of choice, although any Java IDE will work. The feature extraction program reads all of the raw data text files from a directory on the researcher's local disk. One string of data is created from file and stored in a vector. The vector is read in ascending order from index zero to index N, where N is the number of raw data files. A second 245 x N vector containing the identifiable information (i.e., first name, last name, etc.) and the 239 feature values is created for each raw data file [3].

Villani et al. [3] developed fallback trees for single key features and transition features, which have been

implemented in this feature extraction program. Fallback is used to minimize “bad” data caused by a less-than-optimal number of occurrences of a feature. Fallback is implemented by assigning each node on the tree a numeric pair consisting of that feature’s unique numeric identifier as well as its parent’s unique numeric identifier. This allows the programmer to easily change the pairs and thereby changes the structure of the tree.

The feature extraction parameters are listed in Table 1. Parameters three and four, K1 and K2, are for the researcher to optimize the results of the feature extraction. K1 is the minimum number of occurrences of a feature for it to be considered a valid measurement and prevent fallback. K2 is an optimized constant used when taking the weighted average as part of the fallback formula. The fifth parameter (Table 1) is used to enable or disable the fallback functionality. The sixth parameter is optional and available to researchers experimenting with different input text lengths.

Parameter Order	Signifies	Acceptable Values	Explanation of Values
1	Full path to raw data files	C:\RawData	Any path is accepted
2	Outlier Removal	0, R, positive integer J	No outlier removal, recursive outlier removal, J passes of outlier removal
3	K1	Any positive integer	Minimum number of occurrences of a feature
4	K2	Any positive integer	Optimized constant for fallback’s weighted average formula
5	Fallback	0, 1	Do not use fallback, Use fallback
6	Number of raw data keystrokes to consider	Null for all keystrokes, any positive integer. (Program will throw exception if value	Used to optimize the number raw data of keystrokes required to obtain a specific

		exceeds number of key strokes in raw data file.)	level of accuracy. Click counts will always be included.
--	--	--	--

Table 1: Summary of feature extraction parameters.

If the fallback mechanism is enabled and a feature meets the fallback requirements, its value in the feature vector is replaced with the fallback value.

The final process in the feature extractor is to write the contents of the feature vector to a tilde delimited file with N lines (Figure 5).

```

1 Anthony-Visciglia-Submission-24-keyboard-82.5416091954023-76.38232304900181-87.9783
2 07065217391305-62.391304347826086-88.54166666666667-58.513819785645174-43.404979423.
3 0-0-0-0-0-0-0-0-0-0.015332197614991482-0-0-0-0-0-0-0.0068143100511073255-0-0.0187.
4 Anthony-Visciglia-Submission-3-keyboard-75.7016229712859-80.61041904141561-88.28470.
5 65.7164303586322-58.81345565749235-265.611111111111114-51.93593835134344-50.99949828.
6 4-0-0-0-0-0-0-0-0-0-0.0074962518740629685-0.0014992503748125937-0-0-0-0-0-0.
7 Anthony-Visciglia-Submission-5-keyboard-77.92260481497146-80.44183168316832-84.4722.
8 30612244-67.42857142857143-65.625-43.37911323833436-42.338873010696396-53.80540903.
9 5447570332483-0-0-0.005115089514066497-0-0-0-0-0.020460358056265986-0.0076726342710.

```

Figure 5: Example of a feature file showing three input-sample feature vectors.

2.3 Pattern Classifier

The pattern classifier reads from the feature file one line at a time and stores the entries in a matrix (a sequence of feature vectors). The features are then standardized to a value between zero and one. Using the Nearest Neighbor method to compare Euclidean distances, the classification process is either run using a leave-one-out style or, if two feature files are provided, a train-on-one, test-on-another method.

In leave-one-out, the trial entry is logically removed from the feature matrix. The Euclidean distance between the current trial entry and the entries remaining in the feature matrix are calculated and, the entry with the smallest Euclidean distance is the nearest neighbor. If the authors of the training and test entries are the same, a match is declared.

In train-on-one, test-on-another, the Euclidean distance between one entry from the test file’s feature matrix and all entries in the training file’s feature matrix are compared. If the author of the training entry with the smallest Euclidean distance is the same as the test author’s entry, a match is declared.

For both leave-one-out and train-on-one, test-on-another, a tilde delimited text file is created. This file can be imported into a spreadsheet program such as Microsoft® Excel and a confusion matrix is automatically generated (Table 2). This matrix only displays the number of times and percentage of the time the classification program was not confused (the diagonal).

Table 2: Accuracy confusion matrix for 30-subject experiment: subjects identified by initials, first column = number of samples per subject, diagonal entries = percent correct, and off-diagonal entries = percent confused [2].

The researcher has the capability to enable or disable the usage of seven feature areas (Table 3): the duration (single key) averages; duration standard deviations; transition type-1 averages; transition type-1 standard deviations; transition type-2 averages; transition type-2 standard deviations; and percentage features [3].

In addition to classifying using Nearest Neighbor, parameter nine also allows a researcher to use the K-Nearest Neighbor classification method.

Parameter Order	Signifies	Values	Explanation of Values
1	Duration average	Y, N	Use, do not use
2	Duration standard deviation	Y, N	Use, do not use
3	Transition type 1 average	Y, N	Use, do not use
4	Transition type 1 standard deviation	Y, N	Use, do not use
5	Transition type 2 average	Y, N	Use, do not use
6	Transition type 2 standard deviation	Y, N	Use, do not use
7	Percentage features (e.g., input rate, left mouse clicks)	Y, N	Use, do not use

Parameter Order	Signifies	Values	Explanation of Values
8	Experiment style	A, B	Leave-one-out, train/test
9	Classification style	1, an integer greater than 1	Nearest Neighbor or K-Nearest Neighbor where integer indicates majority
10	Match within X attempts	1, an integer greater than 1, not to exceed total number of raw data files	1 = match on first test only 2 = match on test values 1 or 2 – output file indicates “no match!!” for failure to match within N smallest Euclidean distances or “@ #” where # is the index of the smallest Euclidean distance $\leq N$.
11	Test feature file (this determines the number of results)	Path to file	Usually Biofeature++\xyz.features
12 (only if experiment style = B)	Training feature file	Path to file	Usually Biofeature++\xyz.features

Table 3: Parameters for pattern classification program.

3. Implementation Iteration

The accuracy of the system has been scrutinized throughout the development cycle. The initial milestone was matching the calculations generated by the SAS program.

3.1 Fifty-Eight Features

The SAS program used by Bartolacci et al. [1] was established as the standard of correctness. The Java programmers needed to replicate these results before making any of the changes required by the current researcher. Once the feature extractor was implemented, manual testing was begun.

Previous research resulted in a file with the feature values of all raw data files that had been passed through the SAS-based classification program. The Java programmers took one raw data file and manually calculated all of the feature values and feature values after single-pass outlier removal using Microsoft® Excel. These numbers were compared to the results of the 58-feature Java program and the program refined until all values matched. While it was identified that the previous researchers used a single-pass outlier removal, the Java programmers developed a method to recursively remove outliers as well.

3.2 Manual Classification

With confidence in the feature values, the results of the feature extractor were manually classified. The goal was to achieve accuracy of at least 97% when run, using data obtained from the experiment conducted by Curtin et al. [2]. The Java programmers could not achieve this level of accuracy using the program in its current state.

3.3 One Hundred and Twenty-Nine Features

The feature set was increased to 129 features and the process repeated.

3.4 Two Hundred and Thirty-Nine Features

With accuracy still not where it should be, the researcher increased the number of features to its current state of 239 features. These features incorporated keystroke data from across the keyboard as well as the two different transition styles. Lastly, the mouse clicks and “special key” (e.g., home, insert) presses were calculated in a ratio to the whole rather than as single values.

3.5 An Improved System

Using 239 features, recursive outlier removal and the fallback method, the accuracy of the system was increased from roughly 94% to 99.6% when run on data collected by Curtin et al. [2].

4. Development Method

While Extreme Programming’s (XP) pair programming was not used, elements of XP were prevalent in the development process. The programmers held weekly meetings with the client where an updated system was always delivered, critiqued and a new deliverable set for the following week. This meeting was more of a weekly scrum.

5. Object-Oriented Approach

The object-oriented approach to programming was used in both the feature extraction and pattern classification programs.

5.1 An Object-Oriented Feature Extractor

Using classes, a *key feature* class was developed for each feature value. This class encapsulates information with regard to frequency, average, and standard deviation.

The purpose of the Feature Extractor is to iterate over the raw data collection and generate a new collection consisting mainly of averages and standard deviations (the 239 features). The iteration and generation processes are accomplished through the use of two primary classes, *FeatureExtractor* and *Keyfeature*, and a number of secondary derived classes. The *FeatureExtractor* extracts the features from the raw data file and maintains a collection of *KeyFeature objects* that are written to the feature file used by the pattern classification program. The *KeyFeature* is the parent class, which maintains the feature values. It provides methods for accessing and modifying private data members (frequency, average and the standard deviation). It also maintains a collection of its keystrokes and implements the outlier removal and the fallback methods. Depending on the parameter set, outlier removal and/or fallback is done on the data collection. The secondary classes are the individual features. They are represented as a tree [3] for fallback purposes. Each feature has a parent. Among the derived classes are *EKeyFeature* corresponding to the ‘E’ key, and the transition feature class *ALFeature* corresponding to the transition from A to L (Figure 6).

Given the base-derived (inheritance) class relationship, new features can easily be added.

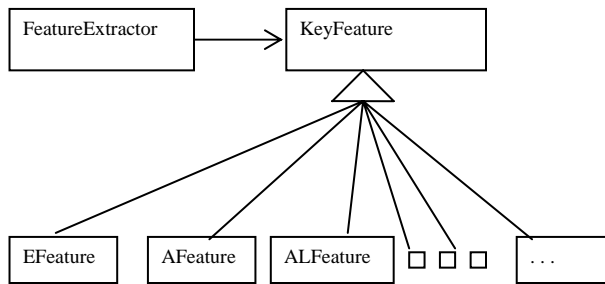


Figure 6: FeatureExtractor Architectural View

[3] M. Villani, C. Tappert, G. Ngo, J. Simone, H. St. Fort and S. Cha, "Keystroke Biometric Recognition Under Ideal and Application-Oriented Conditions," *Proc.- IBC*, IBS, Montreal, Canada; July 2006.

5.2 Object-Oriented Pattern Classification

For reusability, flexibility, and adaptability to changing requirements, the classification system is built on five classes. *KeystrokeProcessor* is the main class which takes the output from the feature extractor and processes the data according to the parameters specified by the user and calls upon its subordinate classes, *Standardize*, *Classifier*, *PrintMatrix*, and *AscendVector*. The purpose of the *Standardize* class is to standardize the data it receives using the formula $x' = x - \min / \max - \min$ so that the data are within a range from 0 to 1. The *PrintMatrix* class is responsible for generating the layout of a confusion matrix in text file. The *Classifier* class determines the nearest match using Euclidean distance between the test data set and each of the training data sets. Lastly, the *AscendVector* class arranges the results of the Euclidean Distance calculations in ascending order to accommodate easy matching of nearest value or nearest K value. Due to their lack of interdependence and clear distinction, the classes can easily be migrated to other systems, additional services added, and/or adapted to accommodate changes in system requirements without major modification.

6. Conclusion

This system has been successfully used producing several accepted conference papers.

References

- [1] Bartolacci, M. Curtin, M. Katzenberg, N. Nwana, S. Cha, and C.C. Tappert, "Applying Keystroke Biometrics for User Verification and Identification," *Proc. MCSCE, MLMTA*, Las Vegas, NV, June 2005.
- [2] M. Curtin, C. Tappert, M. Villani, G. Ngo, J. Simone, H. St. Fort and S. Cha, "Keystroke Biometric Recognition on Long Text Input: A Feasibility Study," *Proc.- IWSCCS*, IMECS, Hong Kong, China; June 2006.