University of Windsor

Project Report - 01

Evolutionary Computing & Cybersecurity

COMP 8920-01 Summer 2019

Project Supervisor: Dr. Sherif Saad

Authors: Mayank Semwal, Vipul Malhotra

1) DATA INSIGHTS:

Typing Speed	• First, dataset is grouped by each column (i.e. password keys) and summation on the basis of subject(users) and sessionIndex (8 sessions for each user). Second, the choice of hold and up-down keys is done to figure out duration of time for which a key is pressed and time elapsed between releasing a key and pressing the following key. • Later, combining the *hold* and *up-down* key values of each subject and session in a single keystroke (31 columns). Finally, computing speed of each user by cumulating average time of *hold* and *up-down* key values of each session (e.g. for user '*s002'* speed is 122.071 seconds)
Data Insights	• Min, Max and Std value of each session for every user to see the variation in their typing behaviour in no. of repetitions (i.e. 50), use this to compare genuine and imposter user in each session. • Correlation Analysis: If *h. period* key is typed so fast it is observed that user takes time to enter the next key and the correlation value is negative. However, if correlation is close to 1 then both the keys are relative to each other.
How Realistic Is the Data	• Sometimes user's mood can affect the typing behaviour which is not considered in the dataset. Diversification of data on the basis of gender, age and nationality is not taken into consideration. • Physical disabilities can result in abrupt typing speed. Dataset collector has ignored typological errors and because of this user has to retype the password therefore in reality while typing password users definitely make significant errors which may affect the behavior analysis.
Limitations	• Firstly, typing patterns can be erratic and inconsistent as something like cramped muscles and sweaty hands can change a person's typing pattern significantly. **Lower Accuracy**: It means low accuracy due to the variations in typing rhythm that caused by external factors such as injury, fatigue, or distraction. **Lower Permanence**: Typing pattern of a user gradually changes as can be noticed in each 50 repetitions, maturing typing proficiency can also be a factor of variations in speed of different users, adaptation to input devices, and other environmental factors (geographic implications).

2) KEYSTROKE DETECTOR: Steps to design and implement anomaly detector.

2.1) Initialization:

- Read the csv file and store it into data frame using pandas.
- Discard all the *down-down* (excluding *return* key) key values because it's the sum of *hold* and *up-down* key values and consider h and ud (i.e. 21 columns).

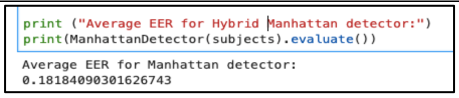
2.2) Evaluation:

- After initialization, pass all unique users into evaluate function.
- Consider current user as genuine and rest as imposters in which genuine user's first 200-time vectors are used for training and next 200 as test samples.
- Detector will take 10 keystroke login samples per user and generate keystroke signature for this user in which imposter set (250 records, 5 per imposter, 50 imposters in all).
- In next step, data will be trained using their mean and std vector are calculated and outliers will be removed by taking into account deviations in typing habits. To reject the outliers first Euclidean distance between training and mean vector is calculated and any vector whose distance is greater than 3 times the standard vector.

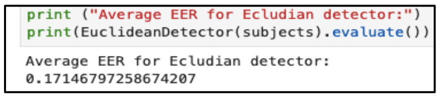
- Pass the test samples, imposter set and results of training function i.e. mean and std values into testing function, it will calculate the Euclidean distance between test sample and mean vector.
- Having obtained the scores for both kind of samples, genuine and imposter, in lists user scores and imposter scores respectively, we evaluate the equal error rate (EER) for the detector.
- Two types of detector we have used: Hybrid Manhattan (used Euclidean dist. to train samples, drop the outliers and test the samples with Manhattan dist.) and Euclidean distance.

2.3) Performance Analysis: In total each user will be tested 700 times (10*50+200) for user's authenticity and will be repeated for all 51 users.

- EER is the intersection point of false rejection rate (FRR) and false acceptance rate (FAR). FRR measures the likelihood that the biometric security system will incorrectly accept an access attempt by an unauthorized user whereas measure of the likelihood that the biometric security system will incorrectly reject an access attempt by an authorized user.
- Results of Manhattan and Euclidian detector



```
print ("Average EER for Hybrid Manhattan detector:")
print(ManhattanDetector(subjects).evaluate())

Average EER for Manhattan detector:
0.18184090301626743
```



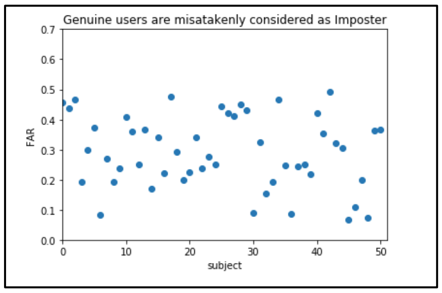
```
print ("Average EER for Ecludian detector:")
print(EuclideanDetector(subjects).evaluate())

Average EER for Ecludian detector:
0.17146797258674207
```

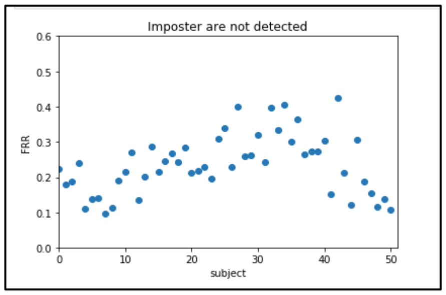
Detector	*Average Equal Error Rate*	*Matching Accuracy*
Hybrid Manhattan(time taking)	0.182	81.8%
Euclidian	0.172	82.8%

3) USER DESCRIPTION:

User Group	*Criteria*	*Description*
Sheep	TPR >= 0.85	If genuine users are able to imitate correctly (true positive rate must be high).
Goat	FRR >= 0.85	User who are prone to false reject or wrongly considered as imposters i.e. FRR (1-TPR) must be high.
Lamb	FAR > 0.45 and FAR<= 0.85	If false acceptance rate is low means the genuine users are easily imitated.
Wolf	FAR >=0.85	If false acceptance rate is high means the imposters are wrongly considered as genuine user.
NOTE: we consider each session of user as the imposter for some other user that is why the value of each user group (sheep, goat, lamb, wolf) is relatively high.		

3.1) User Categorization Analysis Using FAR And FRR





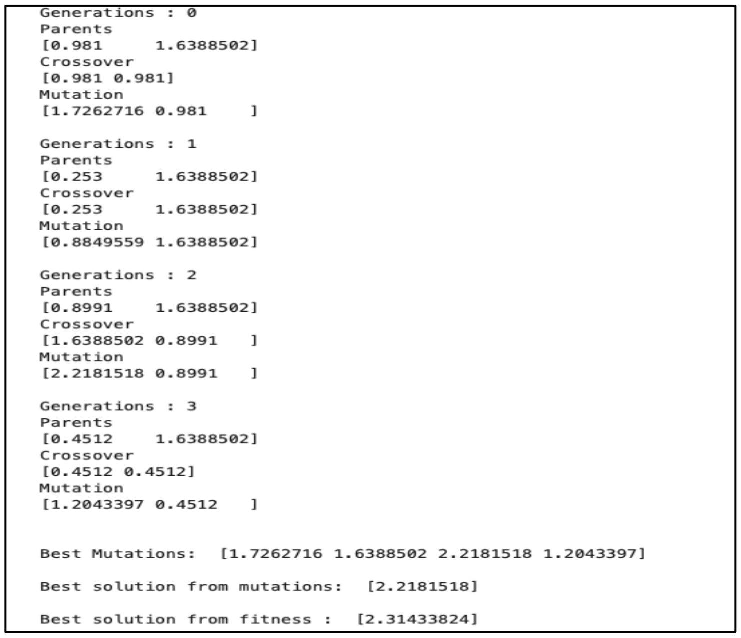
First figure above shows the mean of all users and their False Acceptance Rate with respect to number of users (FAR between 0.1 and 0.45 are genuine, greater than 0.45 are considered as wolf) and second one is

False Rejection rate where genuine users was not able to identify themselves and imposters was also not detected (Users greater than 0.4 are highly unpredictable users, hence consider as goats).

4) BIOMETRIC ADVERSARY FRAMEWORK: Steps to design and implement of Genetic algorithm.

- **Initialization**: We have created an initial population in the shape of an array of fix size and can be any desired size from few individuals to many.
- **Fitness Function**: Each member of the population is evaluated and we calculate a fitness for each individual. The fitness value is calculated using Euclidian distance ((math.sqrt(mean-initialpop)**2)/standard_deviation). If Euclidean distance is greater than or equal to 0.85, value will be appended to fitness array and will be returned.
- **Selection**: It is used to select n number of parents (we have considered two parents). To improve population's overall fitness. We have considered one parent to be randomly selected from fitness array and other parent randomly from the initial population, the basic idea is to diversify selection of parents but also considering single parent to be selected from the best fitted results.
- **Crossover:** During crossover we create new individuals (known as offsprings) by combining aspects of our selected parents. The crossover rate is the probability of acceptance to select individuals of next generation which is 0.5 (i.e. 50%) in our crossover criteria. We have used single point cross-over in which parent to the right of that point are swapped between the two parents. This result in two offsprings each carrying some genetic information from parents.
- **Mutation:** To add a little bit randomness into our population genetics because every combination of solution will be different from initial population. We have used a random value generated in each generation ranging from 0 to 0.5 to be added to offspring.
- **Repeat:** Now, we have our next generation we can start again from fitness until we reach a termination condition.
- **Termination:** If mutation result is greater than or equal to maximum from fitness result array (i.e. fitnessArr.max ()) the generation will break and display the set of best mutations in each generation into an array, out of which we will select the maximum two compare with maximum from fitness array.
- **Goal:** The goal is to get mutation results close to the fitness value.

<u>Output Result</u>:



```
Generations : 0
Parents
[0.981     1.6388502]
Crossover
[0.981 0.981]
Mutation
[1.7262716 0.981    ]

Generations : 1
Parents
[0.253     1.6388502]
Crossover
[0.253     1.6388502]
Mutation
[0.8849559 1.6388502]

Generations : 2
Parents
[0.8991    1.6388502]
Crossover
[1.6388502 0.8991   ]
Mutation
[2.2181518 0.8991   ]

Generations : 3
Parents
[0.4512    1.6388502]
Crossover
[0.4512 0.4512]
Mutation
[1.2043397 0.4512   ]


Best Mutations:  [1.7262716 1.6388502 2.2181518 1.2043397]

Best solution from mutations:  [2.2181518]

Best solution from fitness :  [2.31433824]
```