Models for Keystroke Dynamics Anomaly Detector

Fucheng Zhu
Department of Computer Science
Boston University
Boston, MA 02134
rockfz@bu.edu

Abstract

Taking advantage of the Keystroke bio-metrics information dataset on CMU machine learning website, I developed 4 models to detect an imposter typist from the authenticate user. Alongside developing models, I implemented a program that visualizes and measures the performance of the models using the average AUC after performing prediction on each of the subjects.

1 Introduction

The Keystroke Dynamic is a measurement of timing information collected while someone is typing on a computer keyboard. During the modern remote communications, for example, through an email or messenger, some information is discarded. The hold time, the key choice, whether using left shift key or caps lock key, and even the typos, all these can be information unique to a typists. Much like other biometrics such as voice and blood, Keystroke Dynamics can provide crucial information in tasks of identification and authentication. In this project four models are developed to classify an input user information as whether authentic user or an imposter. This project also aims to compare the performance of the models using some metrics and to reasoning the performance differences.

2 Background

2.1 Data Collection

The creators of the dataset designed a set of apparatus, including a timer and a keyboard to collect typing information from 51 subjects. The 51 subjects were 51 persons recruited in the University. The whole data collection process was split into 8 sessions, each session is on a single day. During each session, each person would type the same password over 50 times, making in total 400 repetitions of Keystroke information per person.

2.2 About Dataset

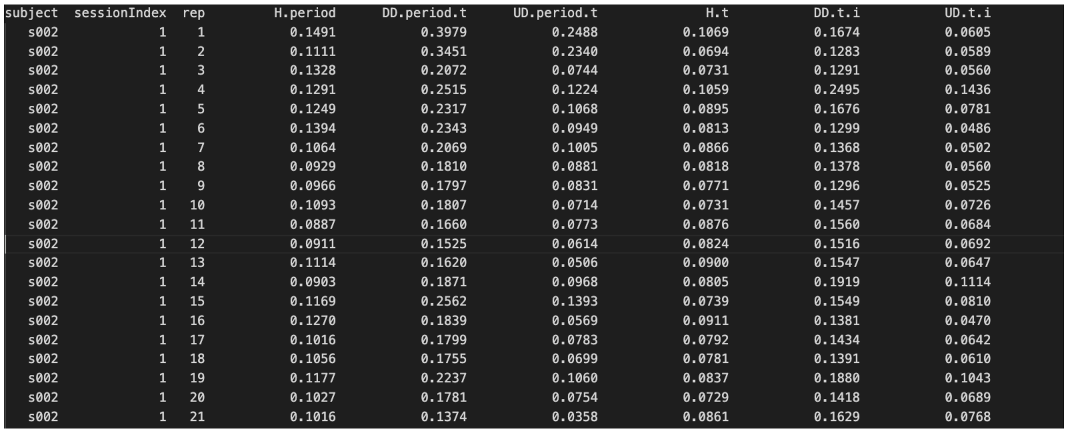
The entire dataset is 20400 * 31. The labels are the subjects' number, ranging from s002 to s057. Each subject participated in 8 session of data collection, contributing to 400 repetitions (400*31). The features are designed to be the press type of a named key, reflected from the column's name. Besides each key's hold time, each digraphs has a keydown-keydown time and a keyup-keydown time. There are in total 31 features.

3 Data Processing

The full dataset is given in an .csv file that is convenient for python project. As shown in the example of figure 1. As a classification problem, the model only needs to return the probability, which is called score here

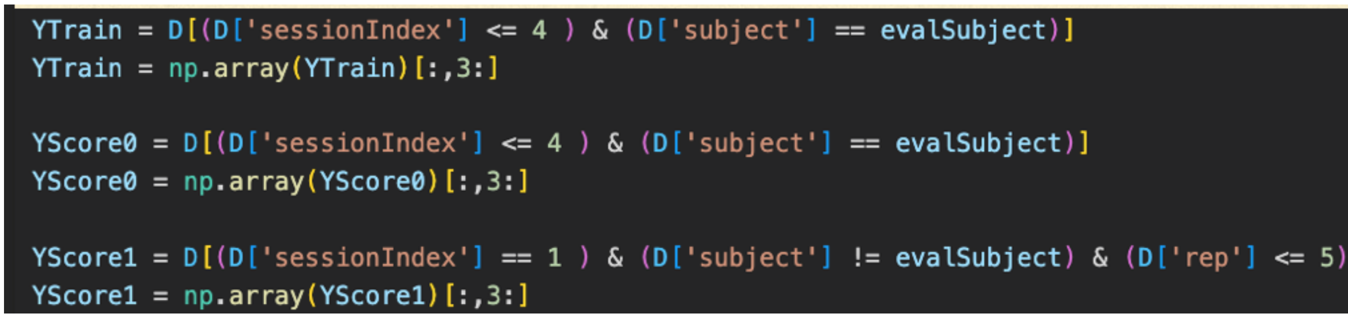
3.1 Data selection

Shown in figure 2. In order to choose TrainingSet, TestingSet, and ImposterSet, I split each subjects' typing information into two halves. The first half, including the first 200 repetitions, is chosen to be the TrainingSet (200 * 31). The second half of repetitions was chosen to be the TestingSet (200 * 31). In this case, since the TrainingSet and TestingSet all comes from the same person, it should all return a value close to zero, as being classified as User input rather than imposter input. Then, I chose every first 5 reps from all other subjects to be the ImposterSet (250 * 31).



subject	sessionIndex	rep	H.period	DD.period.t	UD.period.t	H.t	DD.t.i	UD.t.i
s002	1	1	0.1491	0.3979	0.2488	0.1069	0.1674	0.0605
s002	1	2	0.1111	0.3451	0.2340	0.0694	0.1283	0.0589
s002	1	3	0.1328	0.2072	0.0744	0.0731	0.1291	0.0560
s002	1	4	0.1291	0.2515	0.1224	0.1059	0.2495	0.1436
s002	1	5	0.1249	0.2317	0.1068	0.0895	0.1676	0.0781
s002	1	6	0.1394	0.2343	0.0949	0.0813	0.1299	0.0486
s002	1	7	0.1064	0.2069	0.1005	0.0866	0.1368	0.0502
s002	1	8	0.0929	0.1810	0.0881	0.0818	0.1378	0.0560
s002	1	9	0.0966	0.1797	0.0831	0.0771	0.1296	0.0525
s002	1	10	0.1093	0.1807	0.0714	0.0731	0.1457	0.0726
s002	1	11	0.0887	0.1660	0.0773	0.0876	0.1560	0.0684
s002	1	12	0.0911	0.1525	0.0614	0.0824	0.1516	0.0692
s002	1	13	0.1114	0.1620	0.0506	0.0900	0.1547	0.0647
s002	1	14	0.0903	0.1871	0.0968	0.0805	0.1919	0.1114
s002	1	15	0.1169	0.2562	0.1393	0.0739	0.1549	0.0810
s002	1	16	0.1270	0.1839	0.0569	0.0911	0.1381	0.0470
s002	1	17	0.1016	0.1799	0.0783	0.0792	0.1434	0.0642
s002	1	18	0.1056	0.1755	0.0699	0.0781	0.1391	0.0610
s002	1	19	0.1177	0.2237	0.1060	0.0837	0.1880	0.1043
s002	1	20	0.1027	0.1781	0.0754	0.0729	0.1418	0.0689
s002	1	21	0.1016	0.1374	0.0358	0.0861	0.1629	0.0768

Figure 1: Sample dataset from subject s002



```
YTrain = D[(D['sessionIndex'] <= 4 ) & (D['subject'] == evalSubject)]
YTrain = np.array(YTrain)[:,3:]

YScore0 = D[(D['sessionIndex'] <= 4 ) & (D['subject'] == evalSubject)]
YScore0 = np.array(YScore0)[:,3:]

YScore1 = D[(D['sessionIndex'] == 1 ) & (D['subject'] != evalSubject) & (D['rep'] <= 5)
YScore1 = np.array(YScore1)[:,3:]
```

Figure 2: Splitting dataset

4 The Models

At the model choosing stage, the problem is simplified into a binary classification problem-whether the given new vector is an outlier from the training data or not. Therefore, the spatial distances are employed to detect the outliers. Moreover, the One-class SVM is also a algorithm that performs novelty detection task. The detectors are designed in a two-phase manner, giving convenience to evaluation. During the training phase, the detector takes in the TrainingSet and returns the model. During the scoring phase, the detector takes in the UserSet or the ImposterSet, and returns a score vector. More detailed algorithm will be explained in the implementation section.

4.1 Euclidean Distance Detector

This detector calculates the mean-vector from the TrainingSet, and returns the Euclidean distance of each sample in the input set as the final score using the following formula.

$$d\left(p,q\right)=\sqrt{\sum_{i=1}^{n}\left(q_i-p_i\right)^2}$$

In this case, vector p is the mean-vector, and vector q is each sample vector of the input set. The return score vector is (31 * 1)

4.2 Manhattan Distance Detector

This detector calculates the mean-vector from the TrainingSet, and returns the Manhattan distance of each sample in the input set as the final score using the following formula.

$$d\left(p,q\right)=\sum_{i=1}^{n}\left(|q_i-p_i|\right)$$

p is the mean-vector, and q is each sample input. The return score vector is (31 * 1)

4.3 Mahalanobis Distance Detector

This detector calculates the mean-vector and the inverse of the covariance matrix from the TrainingSet, and returns the Mahalanobis distance of each sample in the input set as the final score using the following formula.

$$d\left(p,q\right)=\sqrt{(q_i-p_i)^T S^{-1}(q_i-p_i)}$$

p is the mean-vector, and q is each sample input. Return vector is (31 * 1)

4.4 One-Class-SVM Detector

The One-class SVM is an unsupervised algorithm that learns a decision function for novelty detection: classifying new data as similar or different to the training set. This detector calls SVM.fit() on the TrainingSet, and returns the probability of each sample in the input set being an outlier as the final score using the following formula. In the scope of this project, I tried to use linear, polynomial, and RBF kernel functions. The return score vector is (31 * 1)

4.5 Neural Network

In this project, NN performed poorly compared to other models. There can be implementations more suitable for this dataset. NN is not included in the program source code.

5 Implementation

5.1 Evaluation Function

Every call to the evaluation function does the following things:
1. The Evaluation function takes in the detector's name and a subject's name, and then it crops the data to generate the following:

YTrain (200*31): the TrainingSet which comes from the first 200 reps of the specified user.
YScore0 (200*31): the TestingSet, the second 200 reps of the same user.
YScore1 (250*31): the ImposterSet which comes from the first 5 reps from all other users.

2. It calls the train(YTrain) method to return a model.
3. Call the scoring function of the detector on YScore0, and get a UserScore (200*31) from YScore0.
4. Call scoring function on YScore1, getting ImposterScore (250*31) from YScore1.
5. Call Performance Metrics Calculation Function on UserScore and ImposterScore.

5.2 Performance Metrics Calculation Function

The function takes in an UserScore and an ImposterScore. It merges the two score vectors and name it the predictions (450*1). Since the first 200 elements are the scores from the same user, it should be smaller compare to the last 250 element. Since this is basically a binary classification problem, it creates a Ygt vector composed of 200 zeros and 250 ones. Then, calculating ROC and AUC for the current subjects performance, returning the AUC.

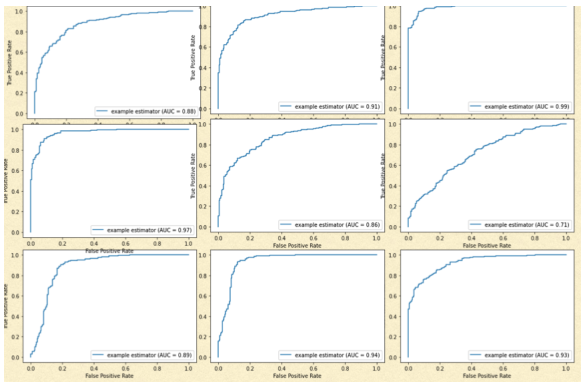


Figure 3: ROC using Manhattan Distance detector from 9 random subjects

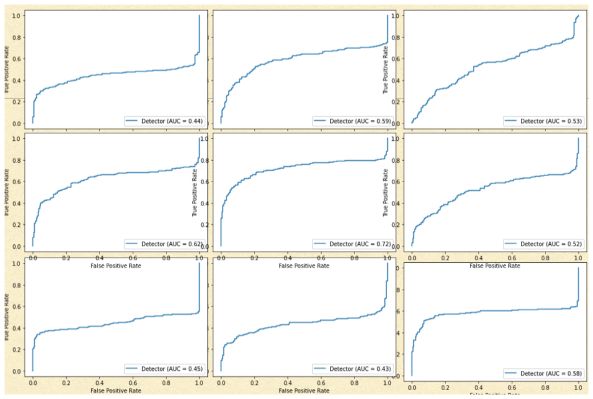


Figure 4: ROC using One-class SVM with RBF kernel from 9 random subjects

5.3 Main Procedure

The main process is that for each detector, choose each subject once, calculate and draw ROC while recording AUC. And for each detector, return the average AUC as the performance metric.

6 Results

Results are shown in table 1.

7 Obervations

SVM as the top performing model is suitable for small and linear data like this. Differences in multiple kernel performances are reflected in the inner relationship of the data. One observation obtained by the better performance in the Statistical models is that statistics gives inferences in the relationships between each variables. Along with the fact that Manhattan distance outperforms Euclidean distance in this anomaly detection is that the features in this dataset is likely to be

Table 1: Average AUC for Detectors

Performance Rank	Detector Name	Average AUC
1	Manhattan Distance Detector	0.9062
2	Euclidean Distance Detector	0.8793
3	SVM w/ linear kernel	0.8590
4	SVM w/ polynomial kernel	0.8583
5	SVM w/ sigmoid kernel	0.8580
6	Manhalanobis Distance Detector	0.8496
7	SVM w/ rbf kernel	0.47
8	Neural Network	< 0.3

proportional to each other. For example, if a person typed one key relatively fast, then it's same likely that he presses other keys for a short time.
In contrast, machine learning algorithms is designed to find the pattern and give the most accurate predictions possible. For a given algorithm, it is not guaranteed that all kinds of dataset could produce satisfying results. In this case, the dataset for a typist can be too small and with too little variations since the person types 50 reps of the same password all at once, where in real life the variation can be much more complicated.

8 Additonal Work

Since the Uniqueness and Universality of Keystroke Dynamics, it can be useful to develop program that collects and analyzes daily typing information to perform anomaly detection tasks. However, due to the large variance in everyone's daily typing pattern, statistic models might not a reliable. Machine learning tools such as Neural Network that deal with more complex data can serve a better role in this field.

References

[1] Kevin S. Killourhy and Roy A. Maxion. "Comparing Anomaly Detectors for Keystroke Dynamics," in Proceedings of the 39th Annual International Conference on Dependable Systems and Networks (DSN-2009), pages 125-134, Estoril, Lisbon, Portugal, June 29-July 2, 2009. IEEE Computer Society Press, Los Alamitos, California, 2009.