ISyE 6740 - Spring 2022
Final Report

Team Member Names: Stacey Franklin, Christopher Lindeman, Alex Riggio

Project Title: User Authentication (Initial and Continuous) Through Keystroke Dynamics

Table of Contents

1 Problem Statement

The world is becoming increasingly digital and as such the number of online accounts per person is greater than ever before. At present, knowledge-based proofs, such as passwords and pins, are the primary method of user authentication and account protection. However, the security they provide can be compromised through simple guessing, social engineering, and brute force techniques. As a result, most people will experience an account breach at some time, and likely more than once. Additionally, even after a valid user has successfully logged into their device, there are situations wherein an unauthorized user could gain access to the device, for example if the device is left unlocked and unattended, or if it is stolen while unlocked.

Biometric authentication methods, which are based on human physiological or behavioral characteristics, offer alternatives that are less susceptible to these threats. Some multi-factor authentication schemes utilize fingerprint scan or iris scan biometric techniques, but both of these are physiological characteristics, not behavioral. In our project, we investigate keystroke dynamics, which refer to the habitual *behavioral* patterns and rhythms that form when a person types on a keyboard or taps on a touchscreen. The authors find it interesting and of note that telegraph operators from the 19th century could identify one another through their unique tapping styles, like a sonic fingerprint. Thus, we seek to answer the question: "Can keystroke dynamics be used an additional line of defense to protect against impersonation attacks during initial authentication as well as continuously post-authentication?"

There has been much research on the topic of keyboard typing styles, but similar research is limited on mobile/touchscreen devices. Given the increasing prevalence of such devices, particularly in corporations and organizations, we seek to study the effectiveness of various machine learning models and statistical techniques learned in this class when applied to the problem of user authentication using touchscreen keying data, herein referred to as "keytap" data, to uncover patterns found in individual users' touchscreen typing behavior.

2 Data Source

The MOBIKEY Keystroke Dynamics Password Database [1] contains data on password typing behavior collected from 54 unique users. Each user generated a minimum of 60 samples of keytap data by entering three different passwords of varying difficulty (easy, logical strong, strong) into a purpose-specific application on a Nexus 7 tablet. Various first-order data features were collected, including features such as key downtime (a.k.a. hold time), time between keytaps, pressure, finger area, etc, and various second-order features were derived from these. (Note that pressure and finger area are not features that can be obtained from traditional keyboard keypress data). Table 2-1. MOBIKEY Dataset Features identifies and describes each feature present in the dataset. It is important to note that all first order features occur multiple times, once for EACH CHARACTER in the password, therefore there are many more features than it seems from the table.

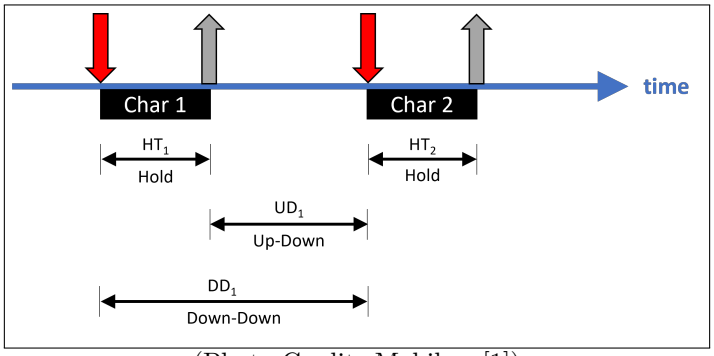
Table 2-1. MOBIKEY Dataset Features

Feature Type	Feature	Description
First order	Hold time for each key (HT)	Time between press and release of a key
First order	Down-down time for each key (DD)	Time between press of key and press of next key
First order	Up-down time for each key (UD)	Time between release of key and press of next key
First order	Key press pressure for each key (P)	
First order	Finger area for each key (FA)	
Second order	Total distance (TD)	Length of the path (in pixels)
Second order	Total Time (TT)	Time needed to type in password
Second order	Mean Velocity (V)	Quotient of TD and TT
Second order	Mean x acceleration (MAX)	Mean acceleration of the device along the x-axis
Second order	Mean y acceleration (MAY)	Mean acceleration of the device along the y-axis
Second order	Mean Z acceleration (MAZ)	Mean acceleration of the device along the z-axis
Second order	Mean hold time (MHT)	
Second order	Mean finger area (MFA)	
Second order	Mean pressure (MP)	

The first order features Hold Time (HT), Down-Down Time (DD), and Up-Down Time (UD) warrant additional explanation.

The first order features exist for EACH character or pairs of consecutive characters in the password. HT exists for each individual character; it is the time that a user spends pressing the character. UD and DD exist for each PAIR of consecutive letters in the password. So, if Char 1 is followed by Char 2 in the password, UD_1 is the time between releasing Char 1 and starting to press Char 2, and DD_1 is the time between starting to press Char 1 and starting to press Char 2. HT_1 is the time spent pressing Char 1, and HT_2 is the time spent pressing Char 2. Figure 2-1. First Order Features illustrates this.

Figure 2-1. First Order Features



(Photo Credit: Mobikey [1])

The MOBIKEY dataset contains samples from entries of 3 different passwords: an 'Easy' password, a 'Logical Strong' password, and a 'Strong' password:

- Easy: kicsikutyatarka
- Logical Strong: Kktsf2!2014
- Strong: .tie5Roanl

Passwords were collected using a custom android application on a Nexus 7 android device. The second order feature 'Total Distance' is the total length of the path of the password as entered. For any given password, the Total Distance for all users will be similar since there will be a maximum distance due to the location of the characters on the on-screen keyboard, but the thought is that individual users will not press each key in exactly the same location as other users, thus the Total Distance for each user should vary slightly, and perhaps enough to be a valuable feature. Figure 2-2. Total Distance (Path Length) shows a screenshot of the app and a yellow line illustrating the 'path' for a particular user entering the 'Easy' password, the distance of the path of which is captured as the 'Total Distance' second order feature.

Figure 2-2. App Screenshot and 'Total Distance' Feature

(Photo Credit: Mobikey [1])

For our purposes however, the authors chose to not consider the 'Easy' password in our research for Initial Authentication use cases. Nor the 'Logical Strong' password. Instead, we chose to focus our efforts on the

'Strong' password, as it is the one most compliant with real-world password complexity requirements present in organizations.

For the Post-Authentication use case, however, it is necessary to architect a simulation of free-text entry data. That is, we will need to randomly stream individual characters into the model in order to simulate free-form post-authentication typing. In order to obtain more data for this specific simulation purpose, we extracted individual first order features for each character from all passwords wherein that character occurred.

3 Methodology

3.1 Scope

As alluded to in the problem statement, we address the following two (2) use cases:

- **Initial authentication -** Granting a user access at the username/password login phase, using keypress data as an additional layer of authentication (effectively a form of multi-factor authentication)
- **Post-authentication -** Performing continuous monitoring of the logged-in user's keytap behavior to determine whether an unauthorized user has gained access to the unlocked device

3.1.1 Models

Herein, we perform a survey of various models. 3.1.1-1. Models Surveyed shows the models and the use case to which they apply.

Table 3.1.1-1. Models Surveyed

Model Name	Desription	Use Case
K-Means Clustering	Unsupervised technique to attempt to distinguish users	Initial Auth
Eigen-Password	A method similar to the "eigenfaces" method	Initial Auth
Lasso	Binary classification (One model for each user)	Initial Auth
Logistic Regression	Binary classification (One model for each user)	Initial Auth
Naive Bayes	Simple Bayesian method for constructing classifiers	Initial Auth
Neural Networks	Both binary classification and a multi-class model	Initial Auth
Random Forest	Averages multiple Decision Trees (weak learners)	Initial Auth
GLR	Mean-drift change detection approach when post-change distribution is unknown	Post-Auth

3.1.2 Feature Selection

For each of the surveyed models, we utilize relevant **feature selection** methods. As stated previously, all first order features listed in Table 1 exist for EVERY character in the password. This therefore yielded a rather large feature set. The authors are curious if the second-order features are necessary at all, since they are essentially derived from the first-order features, or if they essentially perform a form of dimensionality reduction of the full feature set.

Each of the above surveyed methods may use one or more of the following feature selection techniques, as appropriate:

- Wrapper Method: Train each of our models with different combinations of features and select the combinations with the best performance
- Filter Method: Rank features based on ANOVA (F-Statistic), Chi-Squared, or Mutual Information, as appropriate, and select those features above a certain threshold
- Embedded Method: Regression with Lasso performs feature selection as part of the model construction process by driving coefficients to zero

- Principal Component Analysis (PCA): Subsequent dimensionality reduction technique, to be performed after initial feature selection to increase computational efficiency of models, if necessary

3.1.3 Data Preparation (Splitting, Standardization, and Augmentation)

Since our goals are to be able to distinguish different users, we must construct our Training and Test sets such that within a set, there are similar amounts of data points for each user. For multi-class models, such as Decision Trees/Random Forest (Section 3.2.6) and multi-class Neural Networks (Section 3.2.5), where the goal is to train one model that can distinguish individual users, this is achieved simply by a simple train/test split (we used a 70/30 split) due to the fact that the dataset is balanced (roughly the same number of data points for each user). However, in the case of binary classification models, such as Logistic Regression (Section 3.2.3) and the binary classification neural networks (Section 3.2.5), where the goal is to train a single model for each user that will only be able to distinguish between that user and all other users, we cannot simply split the data this way, since the amount of data for the user in question would only be approximately 1/54 that of the amount of data for all other users combined.

Therefore, for binary classification models, we augment the data of the user in question by simply repeatedly randomly sampling from that user's data points, with replacement, until we have sampled enough to be roughly equivalent to the total amount of data for the other 53 users. This way, we will have a roughly equivalent amount of data for our two classes: user and other. Once we have this augmented set, we then do a 70/30 training split.
Once 70/30 training splits are achieved (by either of the two methods above, as appropriate), the training set and the test set are each standardized. Standardization is necessary for this dataset since the numerical values of our features are in very different ranges. Note that the 'label' in our case is the User ID, which does not need to be standardized, since it's categorical.

3.1.4 Cross-Validation and Final Model Evaluation

Unless otherwise specified, $k-$fold cross-validation with $k = 5$ was the cross validation method used.

3.1.5 Evaluation Metrics

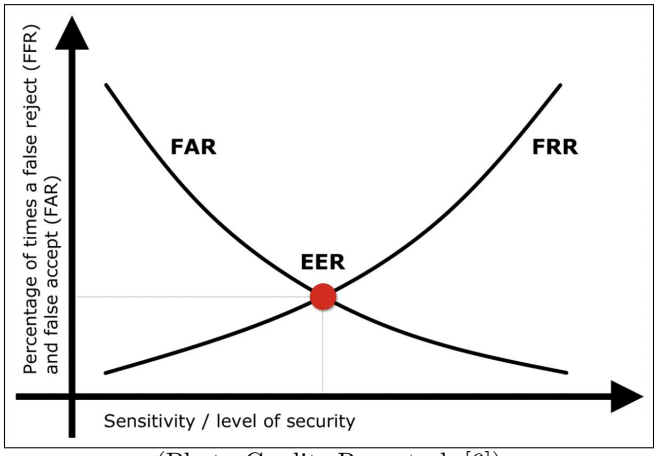
We generate a confusion matrix for all models. However, rather than using the terms 'false positive rate' and 'false negative rate' (which can often be confusing), we use the (security) industry standard terms of 'False Acceptance Rate' (FAR) and 'False Rejection Rate' (FRR), which are easier to understand in this particular use case of user authentication.

- FAR: Percentage of non-authorized individuals allowed access
- FRR: Percentage of authorized individuals rejected access

Depending on the particular security application, reasonable values for the above can range, yet a common convention for FRR is approximately 1%. Higher than that will be noticeable to users and begin to cause an inherent denial of service attack in and of itself. A common convention for FAR is often much lower, getting down to less than 0.001%. However, this is much more flexible based on the nature/use of the security application in question. For us, since this is an imperfect and secondary security measure as part of a much larger overall defense-in-depth organization-level security strategy, we will be willing to accept much higher FAR rates, potentially up to 1%. For our use case, it is more important to not deny access to authorized users. Yet it is important to note that our results herein could easily be tuned according to the security policy of an adopting organization.

Similar to the Crossover Error Rate (CER) in statistics which is where the False Positive and False Negative curves intersect, the analog in the security industry is the Equal Error Rate (EER) which is where the FAR and FRR are equal. Figure 3.1.5-1 illustrates this.

Figure 3.1.5-1. FAR, FAR, and EER



(Photo Credit: Recogtech [6])

The Equal Error Rate (EER) proves useful for comparing model performance, where a lower ERR rate corresponds to better performance.

We will compare the FAR and FRR (and EER, if possible) for all models in the Final Evaluation and Results section at the Conclusion of this paper.

3.2 Initial Authentication

The following subsections discuss each technique in detail, including detailed methodology, feature selection, and analysis of results. All methods will be compared in the Evaluation and Final Results section at the conclusion of this paper.

3.2.1 K-Means Clustering

As K-Means is an unsupervised learning algorithm, our goal was to see if K-Means could distinguish between individual users, or if not, if it could distinguish different groups of similar users.

Parameter Tuning The Silhouette method was chosen as a rigorous and methodical way to choose the optimal parameter k. Unlike the Elbow method, Silhouette does not require a visual interpretation of a plot generated from the computation, and instead, the maximum silhouette coefficient $s(i)$ is chosen[5].

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{max\{a(i,b(i)\}}, & \text{if } |C_i| > 1 \\ 0, & \text{otherwise} \end{cases}$$

For each data point $i \in C_i$ where C_i is the cluster to which i belongs. Let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_I, i \neq j} d(i,j)$$

be the similarity of i to the data points in C_i, and

$$b(i) = min_{i \neq j} \frac{1}{|C_i|} \sum_{j \in C_j} d(i,j)$$

be the average dissimilarity (or minimum average distance) of i to the data points not in C_i. Where $d(i,j)$ is the Euclidean distance between points i and j.

When considering the effectiveness of cluster identification in user authentication, it is imperative that we consider the likelihood of belonging to a particular group as well as the distribution of users in these groups.

For user's belonging to clusters with little representation, the ability to detect unauthorized use may be improved, however the certainty with which we can make the accusation is decreased. For the various values of k, we have the following predictiveness by

$$\frac{\rho(in\_cluster)}{\rho(out\_of\_cluster)}$$

Table 3.2.1-1. Distribution of users per cluster for various values of k

K	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6
2	2057	1246				
3	1221	1203	879			
4	1155	988	844	316		
5	1129	730	654	605	185	
6	847	652	594	581	451	178

Results Despite the optimal parameter selected by the silhouette method, further investigation into the distribution of cluster sizes has led us to choose $k = 6$ as a cluster that seems closest to an even distribution. This allows us to identify a user as belonging to a cluster. The probability of belonging to a particular cluster is $\frac{1}{6}$ on average, and the greatest probability of a false acceptance of nearly 0.95 given by

$$max\left\{\frac{|C_i|}{\sum_{j=1}^{k}|C_{i_j}|}\right\}, \forall C_i$$

As we see from Table 1, the distribution of users per cluster is equally distributed for many of the clusters up to $k = 6$, which means that using K-means clustering alone to predict a user's likelihood of belonging may be ineffective.

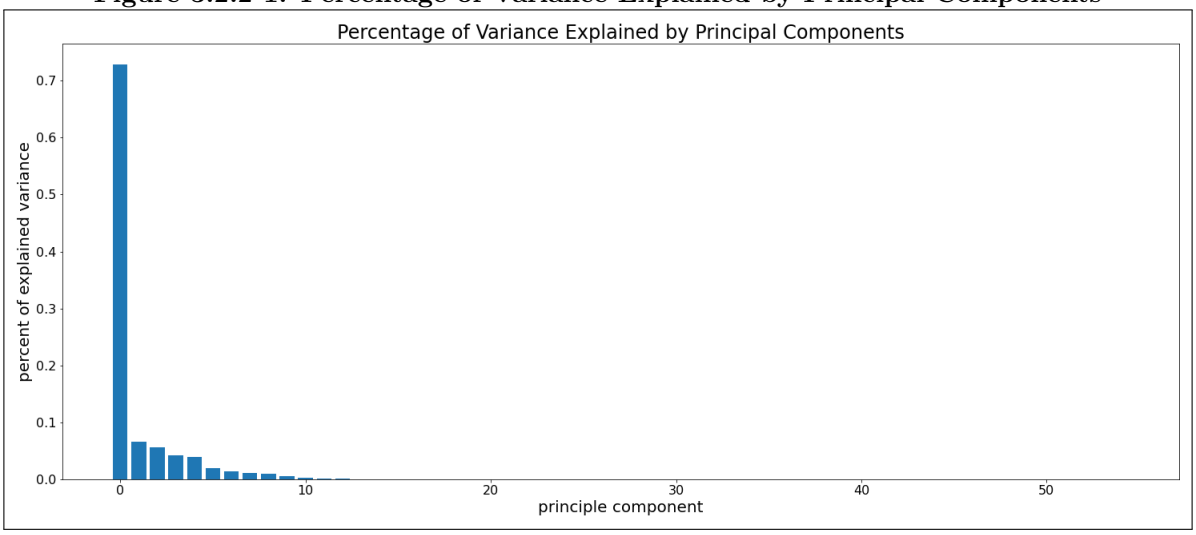
By combining the clusters 5 and 6, we could improve the capability of this method to report a significantly, though this need for human intervention may defeat the purpose of the cost-saving measures of utilizing such a simple method as K-means. Even under the assumption that employees are a representative sample of the population, using K-means has significant issues regarding the trade-off of the accuracy and complexity for various k values. While running a fixed number t of iterations makes K-means no longer an NP-hard problem, the $O(tknd)$ time complexity grows significantly with the increased k. Using the sklearn package, with default number of iterations 300, even our small data set of 54 users increases the time complexity for the algorithm by $71,344,800$ with each increase of k. Feature selection could improve this time complexity significantly.

For all of these reasons, we have determined that K-means is an ineffective method for identifying a user in any reasonable fashion. Even by implementing feature selection, the effectiveness of in-group vs. out-of-group analysis would be highly limited at best.

3.2.2 Eigen-Password

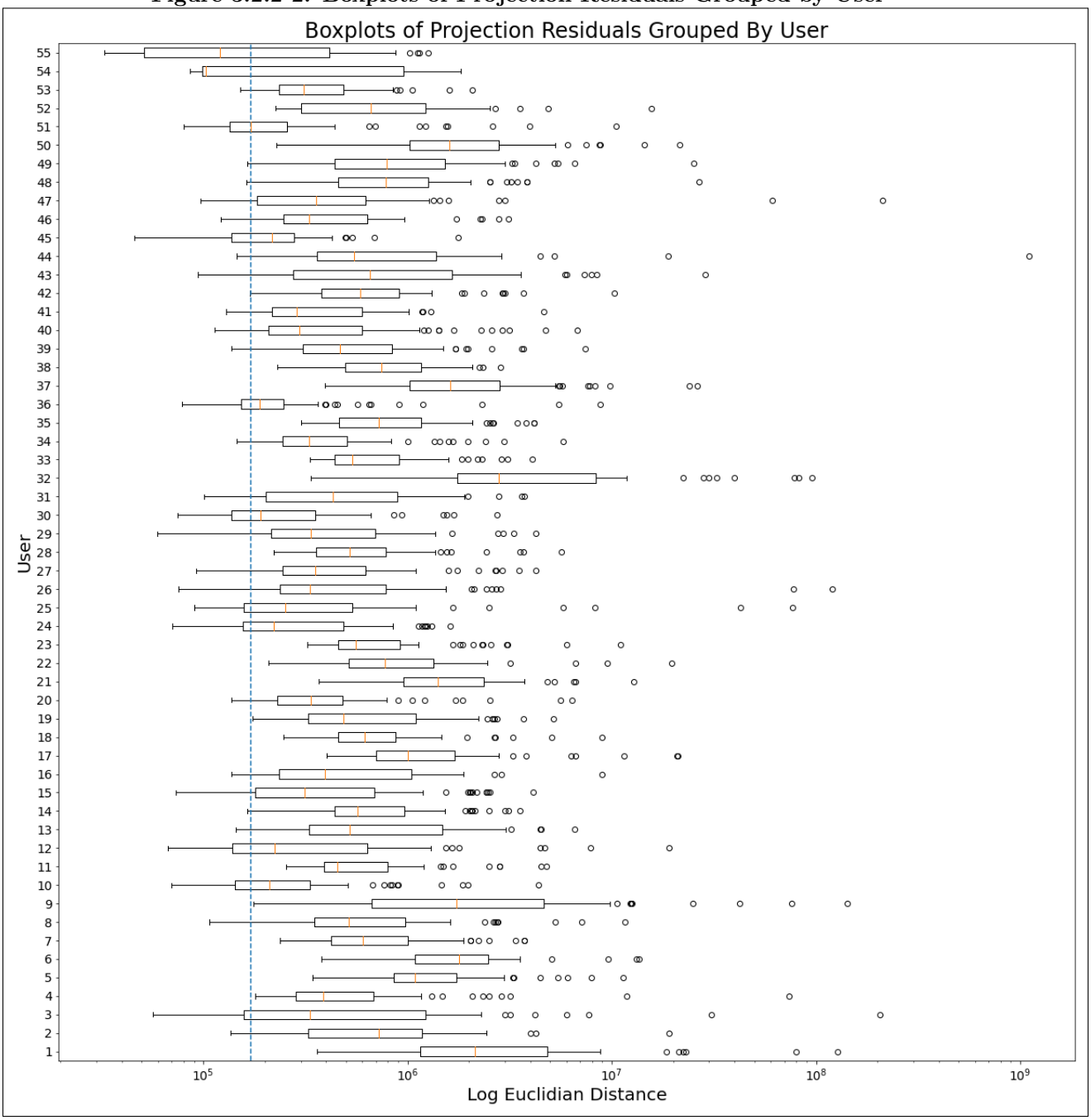
We were interested in exploring whether the projection residual method we learned in class was a viable strategy for identifying authorized users from unauthorized users. To start, we defined the "training" set as 90% of all authorized-user samples and designated the other 10% for testing, along with all samples of the unauthorized users. We then calculated the mean of the training set and centered both the training set and test set using this mean. We then used the function linalg.svd() from the numpy package to perform singular value decomposition on the centered training set. The resulting singular values were squared and each divided by their collective sum to determine the percentage of variance explained in each of the eigenvectors found in the matrix V transpose (Figure 3.2.2-1).

Figure 3.2.2-1. Percentage of Variance Explained by Principal Components



We decided to project our data on the first two principal components since they explained close to 80% of the variance in the data. With our data projected we then calculated the projection residual for each sample as the squared euclidean distance of the projected sample from the original sample. A preliminary threshold was then visually chosen based on boxplots of the projection residuals for the different authorized users. We decided a reasonable threshold was at the 60^{th} percentile of the authorized-user training samples (Figure 3.2.2-2). Our reasoning was that the actual authorized user would on average be denied 40% of the time. However, if prompted to log-in up to 4 times, the chances of them being rejected all four times would drop to around 2.5%, which of course is in no way ideal but given the limitations of the model was a compromise.

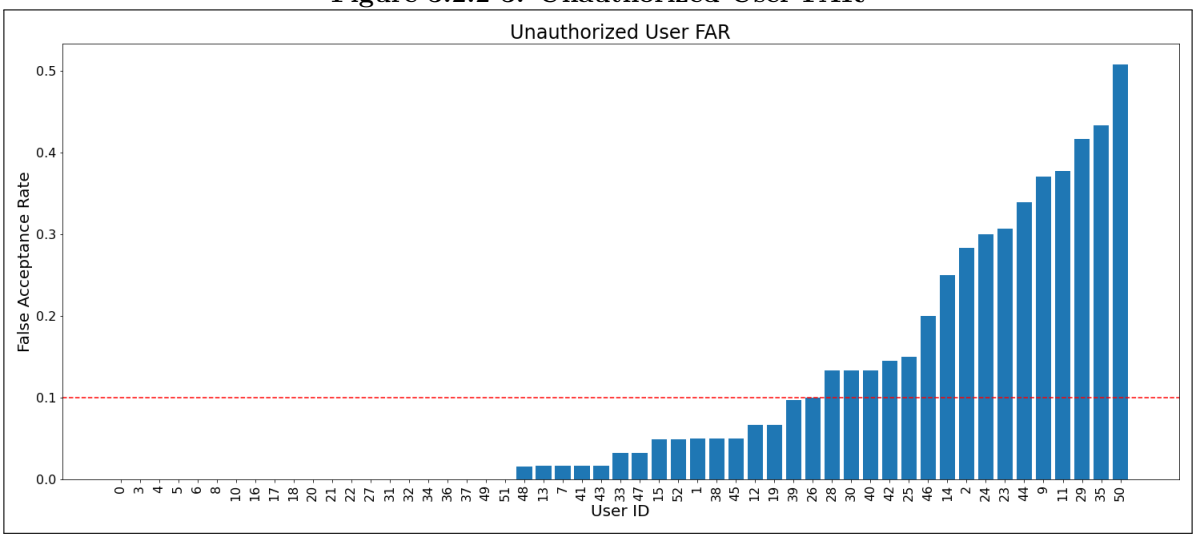
Figure 3.2.2-2. Boxplots of Projection Residuals Grouped by User



Boxplots of the projection residuals for the authorized-user training samples (y-axis = 55), the authorized-user testing samples (y-axis=54), and all other unauthorized users (y-axis= 1 to 53). Dashed blue line is the designated threshold at the 50th percentile of authorized-user training samples (y-axis = 55).

On the other hand, we found that close to 68% of unauthorized users had FARs of less than 10% (Figure 3.2.2-3). However, some unauthorized users had FARs close to 50%. There are definitely weaknesses with this method, but it may have some use as a secondary safety measure that raises flags in the overall security system. We do not recommend it be used as a first line of defense where access is granted or denied based purely off the results of the model.

Figure 3.2.2-3. Unauthorized User FAR



False Acceptance Rates for each other unauthorized user with authorized-user 20 as the reference. Dashed red line delineates a 10% FAR.

It is important to note that the results presented here were just for one authorized user, so it is likely some authorized users will have better separation and others will not. Because our preliminary results were less promising than other methods we had proposed, we decided to end our investigation here to devote more effort elsewhere. Projection residuals are still an interesting methodology worth exploring and further adjustments may be able to be made that can improve performance.

3.2.3 Logistic Regression

Logistic regression models the relationship between predictor variables and a categorical response variable. Given the predictors, the probability of success is defined as:

$$Pr(Y_i = 1|X_i) = \frac{exp(\alpha + \sum_{i=1}^{d} \beta_i X_i)}{1 + exp(\alpha + \sum_{i=1}^{d} \beta_i X_i)}$$

Samples are assigned class membership if the probability of success is greater than or equal to a defined threshold. For example, if the threshold is 0.5, then any sample with a probability of 0.5 or more will be assigned to that class, and any sample less than 0.5 will not.

For this project we framed the problem as there being one authorized user per account/device and any other person attempting to log-in would be defined as an unauthorized user that should be denied access. Therefore, the aim of our model was to be able to identify authorized users from unauthorized users. To accomplish this we trained a different binary logistic regression model for each authorized user. The data for each model consisted of two labels, the authorized user and all other 53 unauthorized users combined. However, this led to a class imbalance since there were approximately 60 samples per user, meaning one class would have 60 samples and the other would have around 53 times that. To avoid the associated bias of such an imbalance, we implemented both up-sampling of the smaller authorized-user class and down-sampling of the unauthorized-user class. Ideally, we would have liked to have only up-sampled the authorized-user class to match the number of unauthorized-user samples. However, in doing so we experienced unreasonable convergence times with our models, an issue we were unable to troubleshoot. Therefore, to overcome this obstacle, we up-sampled the authorized group as much as we could while still maintaining reasonable convergence times. This ended up bringing the authorized-user group to 7 times its original size, approximately 420 samples, and down-sampling the non-authorized group to match.

LASSO Feature Selection adds an L1 penalty term to the log likelihood function as:

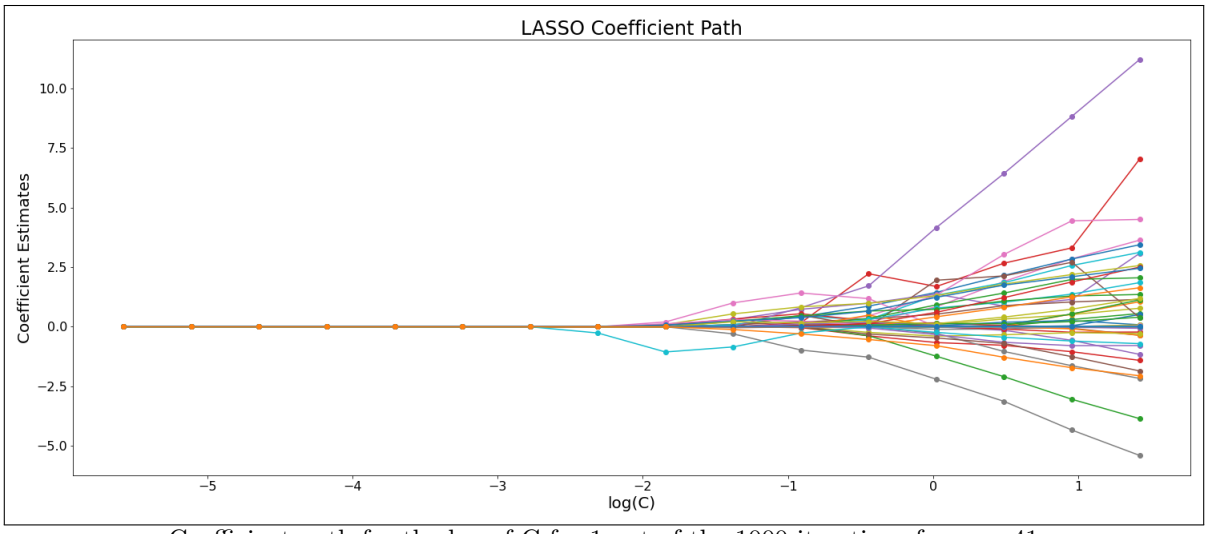
$$(Y - X\beta)^T(Y - X\beta) + \lambda|\beta|_1 \tag{1}$$

Where $|\beta|_1 = \sum_{j=1}^{p} |\beta|_j$. λ is selected such that the resulting model minimizes the error on the test set.

We decided on LASSO for feature selection because it was a unique strategy available to Regression that tends to work well when there are a limited number of samples and a high number of features. By reducing the magnitude of the coefficients, shrinking a portion to zero, it has the potential to lower variance and bias.

Our custom implementation of a cross-validation like procedure involved 1000 iterations using the data set that we had augmented through up- and down-sampling. For each iteration we divided the augmented data set into an 80% split for the training set and a 20% split for the validation set. Next we performed LASSO logistic regression using the LogisticRegression() function in the sklearn package with the arguments penalty and solver set to "l1" and "liblinear", respectively. For the argument C, we looped through the log-space of 0 to 7 multiplied by the lowest bound for C where the model is guaranteed not to be empty as derived by the l1_min_c() function in the sklearn package. In this loop, the estimated coefficients were then selected and recorded for the model with the highest accuracy score using a threshold of 0.5. From these estimated coefficients, we were only interested in the features for which the estimated coefficients were greater than zero (Figure 3.2.3-1).

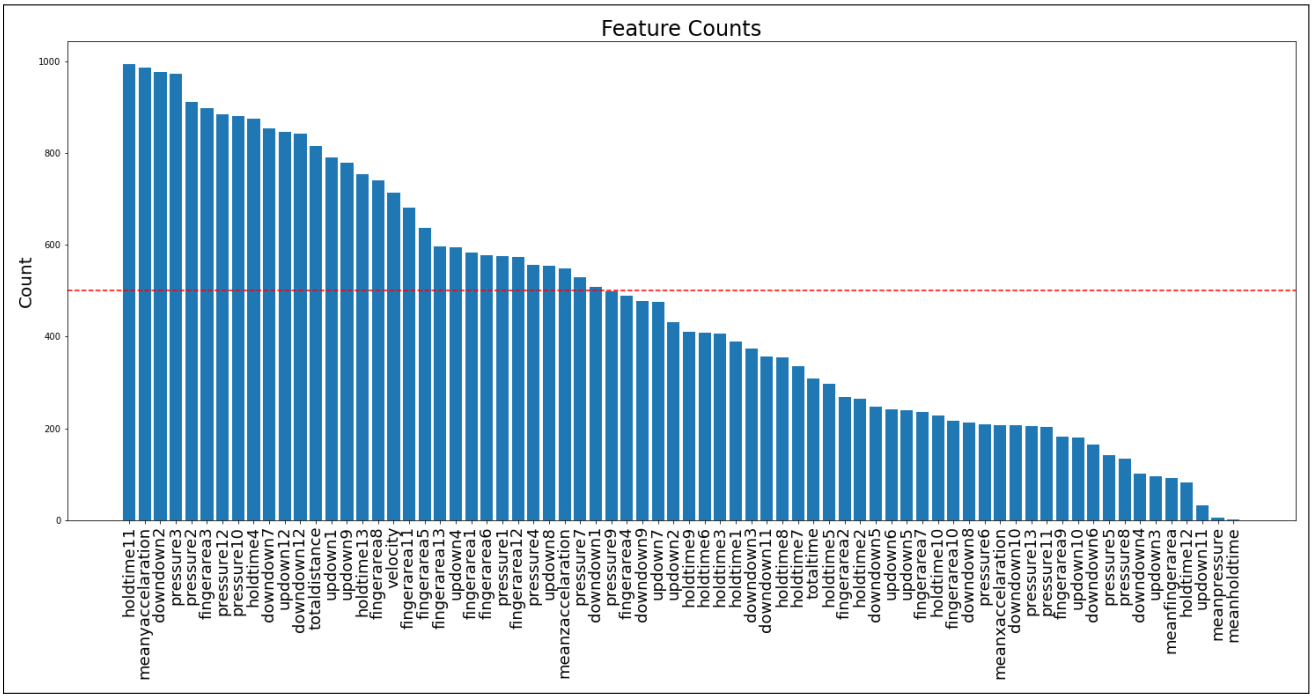
Figure 3.2.3-1. LASSO Coefficient Path



Coefficient path for the log of C for 1 out of the 1000 iterations for user 41

At the end of the 1000 iterations, we had 1000 sets of features. The counts of these features were totaled, and we decided to select the features that were found in at least half of the iterations (counts greater than or equal to 500). For authorized-user 41 there were 32 features in total selected from the original 72 features, providing an approximate 56% reduction in size of the feature space (Figure 3.2.3-2). Keep in mind this process must be done for each authorized-user as each authorized-user requires a unique final model that can identify whether the person logging into their account/device is them or not.

Figure 3.2.3-2. Feature Counts



Counts of features with positive coefficients from models with the highest accuracy in 1000 trials of LASSO logistic regression.

Threshold Tuning and Model Validation With our selected features, we then performed five-fold cross validation splitting 70% of the augmented data set into 5 equal sized subsets. For each of the 5 iterations we used a distinct subset as the validation set and combined the other subsets to form the training set. Within each fold we then looped through 50 evenly spaced threshold values that spanned the interval 0 to 1. At each threshold value we recorded the absolute difference between the FAR and FRR scores. Averaging across the 5 iterations for each threshold, we then selected the threshold with the smallest absolute difference between the FAR and FRR scores (approximate ERR). Interestingly, we found that the prediction probabilities were so absolute, virtually always 0 or 1, that adjusting the threshold had very little impact on the classification of the samples (Table 3.2.3-1). Therefore, for such instances, we suggest choosing a threshold of 0.5 as it might be the safest choice in the event the model responds differently to new, incoming data.

Table 3.2.3-1. Scoring Measures

Threshold	0.0	0.02	0.04	0.06	0.08	.10	...	0.90	0.92	0.94	0.96	0.98	1.00
Mean Absolute FAR/FRR	0.50	0.50	0.49	0.49	0.49	.48	...	0.47	0.47	0.48	0.48	0.49	0.48
Mean Accuracy	0.50	1.00	1.00	1.00	1.00	1.00	...	1.00	1.00	1.00	1.00	1.00	0.99

Table 1: Mean absolute FAR/FRR differences and accuracy for authorized-user 41.

Model Performance For each authorized-user model we measured performance by training the model using the 70% subset of data with the reduced feature space used in the threshold tuning and model validation stage. We then used the other 30% subset of data for testing, and recorded the FAR, FRR, and accuracy scores. To get an overall measure of performance of logistic regression for identifying authorized users from unauthorized users, we then averaged the FAR, FRR, and accuracy scores across all the authorized-user specific models.

3.2.4 Naive Bayes

About The Naïve Bayes classifier uses the relative frequencies of features and classes to produce a proportionality of y given the data by the relation

$$\begin{aligned} P(y_i|x_j) &= P(x_j|y_i) * P(y_i), \forall j \in n \\ &= P(x_1|y_i) * P(x_2|y_i) * P(x_3|y_i) * ...P(x_n|y_i) * P(y_i) \end{aligned}$$

It follows from the above equation that the x_is are assumed to be independent. While we have no expertise in the field, we can assume that many of our datum are related, such as finger area and pressure (heavy fingers are wider), or x,y, or z acceleration (some users just generally move more than others).

Independence Testing In order to properly assess the usefulness of the Naïve Bayes classifier, we test the independence of the predictors, using a χ^2 test. This can be employed on non-negative features, so for our purposes, we will remove any features that involve negative numbers. These are just the mean acceleration values, leaving us with 69 columns to test. Performing a χ^2 test for independence, 27 values are returned as independent. Mutual information for the continuous random variables, given by

$$I(X;Y) = \int_y \int_x P_{(X,Y)}(x,y) log \left(\frac{P_{(X,Y)}(x,y)}{P_X(x)P_Y(y)} \right) dxdy$$

is used, as we have some discrete data, but include continuous data, returns scores that identify significant dependence for all predictors.
We cannot visualize independence, and while not sufficient to identify independence on its own, we can look at the linear correlations of the pairs of variables. This is but a small subset of the many pairs of variables to consider, and their independence is required for this classifier to work properly. We can see that several of the variables for hold times show strong linear correlation, which may affect the ability of the Naïve Bayes classifier to achieve acceptable accuracy.

Results Fit on the raw data, a Gaussian Naïve Bayes classifier had just 69.89% accuracy. In an effort to explore our options, we performed one Gaussian Naïve Bayes classification on the subset of columns referenced by our χ^2 results. This resulted in an accuracy of just 50%. As such, it appear that mutual information has been a more appropriate indicator of the dependence of the variables with regard to their effects on Gaussian Naïve Bayes classification, and that the lack of independence in the features has led to unacceptable accuracy in the Naïve Bayes model. The insight from the pair plot proved very useful (Figure 3.2.4-1), as a cursory look at the correlation matrix showed no particular reason for concern. On closer inspection, we do find some strong negative correlations, which again may be an indication of our suspected dependence. (Figure 3.2.4-2)

Figure 3.2.4-1. Pair Plot of 'holdtimeX' features

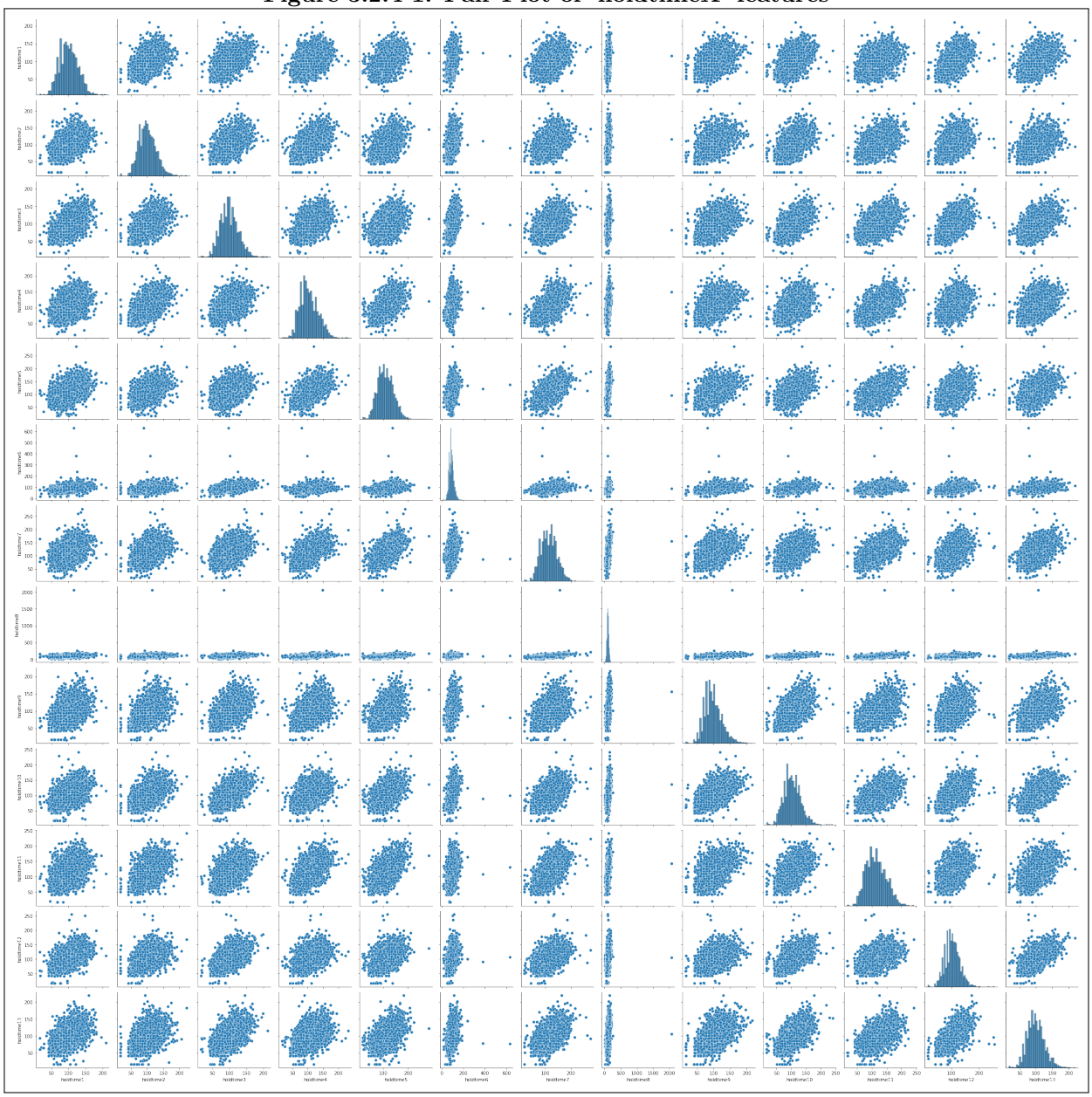
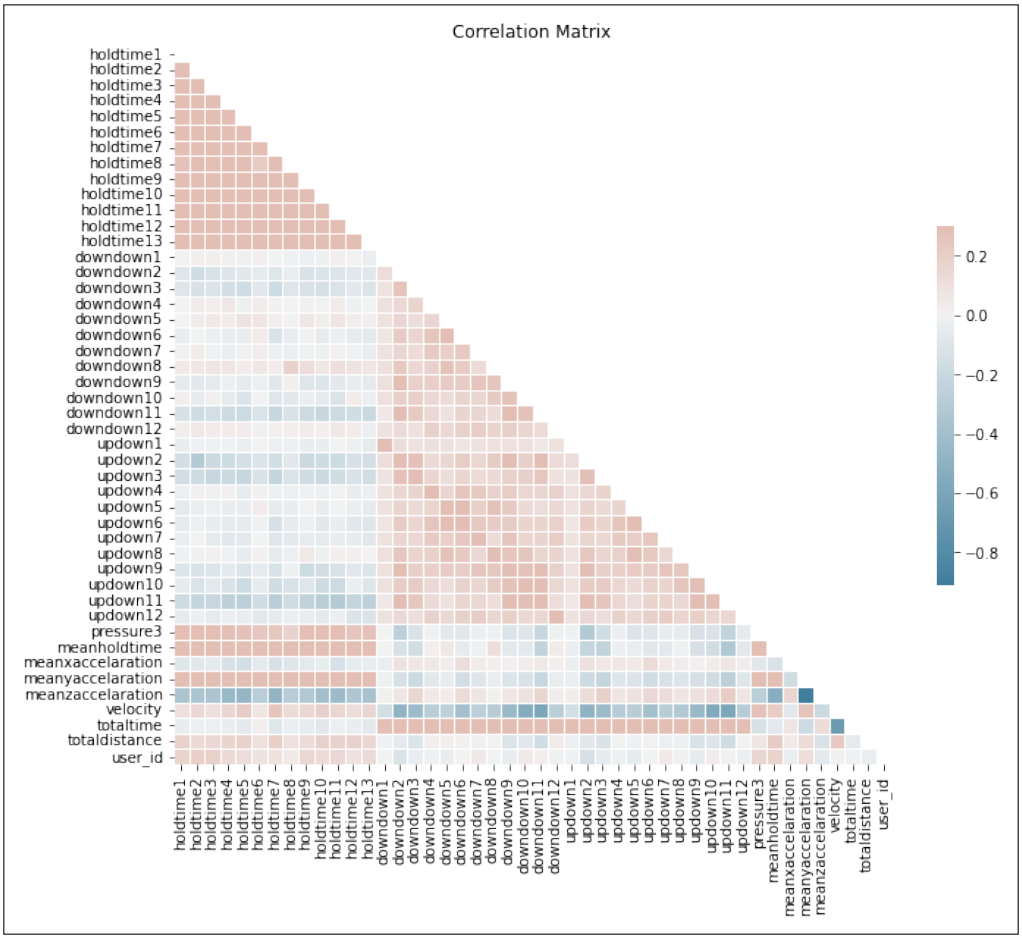


Figure 3.2.4-2. Complete Correlation matrix



3.2.5 Neural Networks

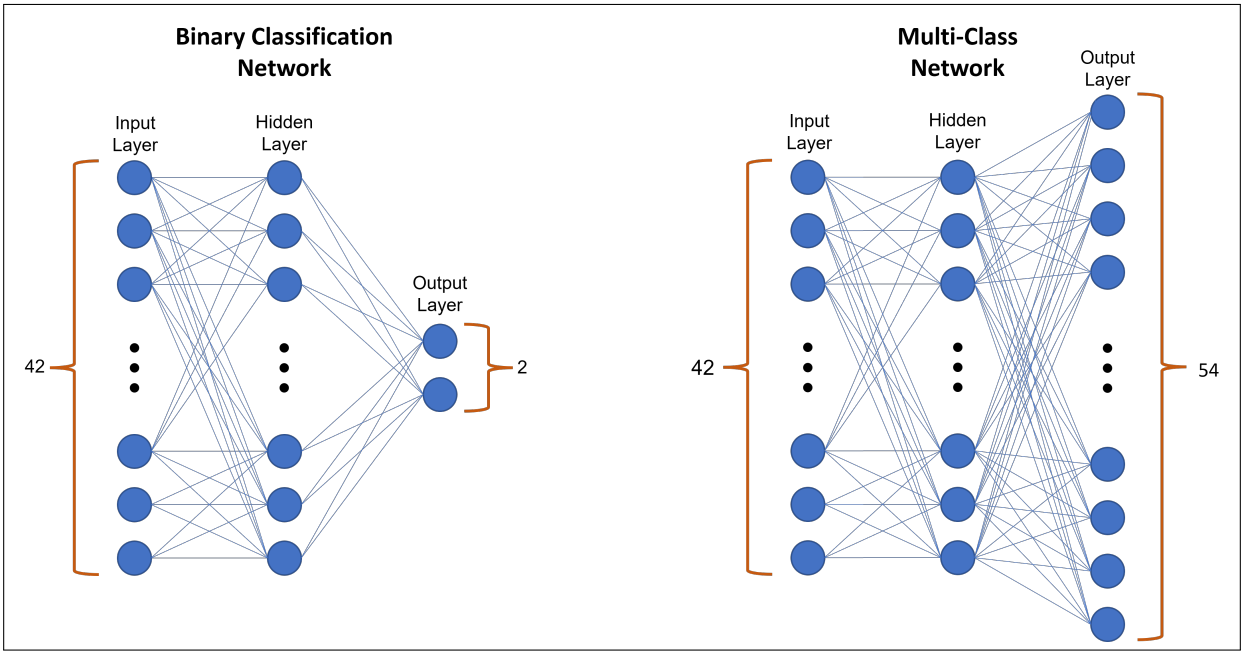
For Neural Networks, we investigate training both a binary classification network for each user, as well as a multi-class network for all users as a whole. Figure 3.2.5-1 shows both these architectures.

Both network architectures will have the same number of nodes in their input layers, since the number of input nodes is determined by the number of features in the dataset. These architectures show 42 features, since our feature selection phase down-selects 42 features out of the original 72. (Refer to Feature Selection subsection below.)

For the output layer, the binary classification network will be trained for each individual user, the goal of which is to identify whether the input data point corresponds to the user in question, or someone else. For example, consider the binary classification network trained for User 100; the output layer will have two nodes corresponding to 'User 100' or 'Other'. Whereas the multi-class network will have 54 output nodes, one corresponding to each user.

We determine the best choice for the number of hidden layers, by performing a cross-validation exercise. (Refer to Cross-Validation subsection below.)

Figure 3.2.5-1. Neural Network Architectures Considered



According to the 'Cost of Insider Threats: Global Report' [4], insider threat incidents have risen 44% over the past two years, with costs per incident up more than a third to $15.38 million. Thus, being able to identify a potential insider threat could be a valuable asset to organizations. We explore the idea that a Multi-Class Neural Network model could potentially identify an insider. Consider the scenario wherein an insider attacker logs into another user's device using the correct password (nefariously obtained in some way). The multi-class Neural Network model could not only identify that the person entering the password is *not* the correct user (and therefore deny access to the device), but could *also* identify the user that the attacker most closely resembles. Such a technique could not be sufficient evidence to warrant dismissal of the user, but it could be used as part of an overall defense-in-depth security strategy wherein if a user was flagged as having potentially been logging into other's devices (beyond what would be considered within the realm of normal model error), the organization's security team could then further investigate the user. The incident response policy can obviously be tailored for each organization.

The idea behind the use of Binary Classification Neural Networks (one per user) is more straight-forward. Train a model for each user, and load the model onto that user's device. The model would evaluate whether the person entering the password on the device was indeed the correct user. If so, allow access. If not, deny entry.

Feature Selection

The first step is to perform feature selection. Neural Networks are computationally intensive to train, and we do not have many data points in our dataset, yet we have 72 features. So we need to find a subset of features that will still achieve reasonable accuracy.

To this end, we perform a loop of multiple iterations, wherein each iteration is performed on a different value of k = number of features. For example, the first loop selects the best set of $k = 1$ feature, the 2nd loop selects the best set of $k = 2$ features, and so on, all the way up to $k = 72$ features. Within each iteration of the loop, we utilize scikit-learn's 'SelectKBest' function to select the k best features according to ANOVA (Analysis of Variance), which uses the F-statistic under the hood to determine whether individual features are independent from the target variable. The F-Statistic (or F-Test) calculates the ratio between two variance values. It is important to note that ANOVA is commonly used when one variable is numeric and one is categorical [5], such as is our case here, since our feature data is numerical continuous, whereas our label data (user_id) is categorical. We pass in a default Neural Network model by which to perform the computation.

Figure 3.2.5-2 shows the plot of accuracy for each value of k = number of features. 42 was selected as the best choice, as this resulted in a nearly equivalent accuracy to when all features were used. As our use case is a matter of security, high accuracy is important, so we want the minimum features possible that will achieve high accuracy. A different use case might select $\tilde{3}5$ features and feel that is still sufficient accuracy, but for us, we err on the side of security and opt to select a few more features.

Figure 3.2.5-3 shows the features identified by 'SelectKBest' of 42 features.

Figure 3.2.5-2. Neural Network Feature Selection (SelectKBest w/ ANOVA)

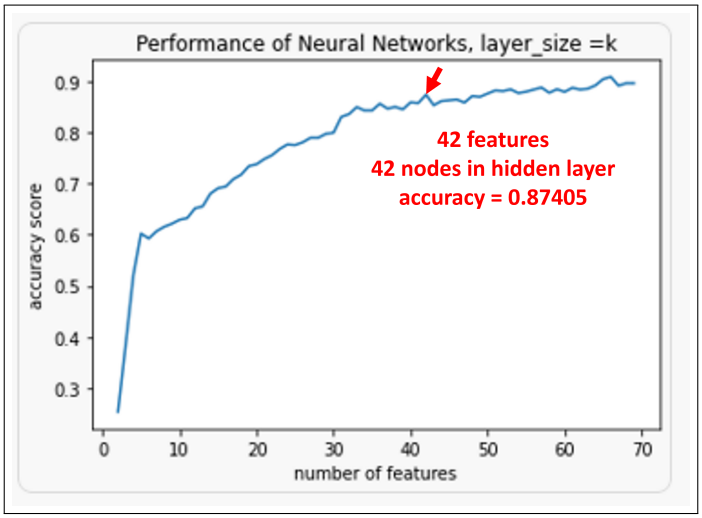


Figure 3.2.5-2. Neural Network Feature Selection (SelectKBest w/ ANOVA)

```
"['holdtime1', 'holdtime2', 'holdtime3', 'holdtime4', 'holdtime5', 'holdtime6', 'holdtime7', 'holdtime9', 'holdtime10',
'holdtime11', 'holdtime12', 'holdtime13', 'downdown2', 'downdown11', 'updown2', 'updown11', 'pressure1', 'pressure2',
'pressure3', 'pressure4', 'pressure5', 'pressure6', 'pressure7', 'pressure8', 'pressure9', 'pressure10', 'pressure11',
'pressure12', 'pressure13', 'fingerarea2', 'fingerarea3', 'fingerarea6', 'fingerarea7', 'meanholdtime', 'meanpressure',
'meanfingerarea', 'meanxaccelaration', 'meanyaccelaration', 'meanzaccelaration', 'velocity', 'totaltime', 'totaldistance']"
```

Recalling that the password is '.tieRoanl', one observation that can be made from this result is that the time *between* key taps doesn't seem to be that important, save for between characters 2 and 3 (downdown2, updown2) and between characters 11 and 12 (downdown11, updown11). Nearly all hold times and pressure features are selected, but only two (2) of the downdown and updown features. Thus, it seems that features corresponding to unique keys and not pairs of keys, end up being more relevant. We also observe that the fingerarea features selected correspond to keys in the top row of a QWERTY keyboard. Perhaps the fact that a user's finger (likely the thumb) must be at a more obtuse angle when touching those characters accounts for more of a variation in finger area between users.

We now utilize this down-selected set of features for both our multi-class and binary classification neural networks.

Multi-Class Network Cross Validation

The next step is to tune our model parameters. We start with the Multi-Class neural network. For our model, we use scikit-learn's MLPClassifier. The parameters of importance for our purposes are:

- hidden_layer_sizes: **Requires tuning** - Specifies the number of hidden layers and the number of neurons in each layer; we try all values between the number of features (nodes in input layer) and the number of unique users (nodes in output layer), i.e., all numbers in the range [42, 54].

- activation: Specifies the activation function to use, defaults to ReLu, which is acceptable for our purposes, as it has been shown to outperform Sigmoid
- solver: We set this to 'sgd' (stochastic gradient descent)
- learning_rate: We set this to be 'adaptive', which keeps the learning rate constant as long as training loss keeps decreasing, otherwise it will decrease the learning rate
- learning_rate_init: We accept the default of 0.001. (We tried a few different learning rates, but none made any difference at all, so we ceased to consider this parameter as one that required tuning, and focused our cross-validation efforts on tuning the hidden layer size)

We performed 5-fold cross-validation to tune the hidden_layer_size parameter. Figure 3.2.5-3 shows the results of the cross validation, where 'Run i' is the loop performed with fold i as the test set.

Figure 3.2.5-3. Cross Validation Results

	Run 1 Best k = 46	Run 2 Best k = 48	Run 3 Best k = 53	Run 4 Best k = 54	Run 5 Best k = 53
Nodes: 42	accuracy: 0.841	accuracy: 0.863	accuracy: 0.861	accuracy: 0.857	accuracy: 0.848
Nodes: 43	accuracy: 0.858	accuracy: 0.851	accuracy: 0.854	accuracy: 0.853	accuracy: 0.853
Nodes: 44	accuracy: 0.862	accuracy: 0.859	accuracy: 0.867	accuracy: 0.865	accuracy: 0.859
Nodes: 45	accuracy: 0.856	accuracy: 0.864	accuracy: 0.854	accuracy: 0.854	accuracy: 0.853
Nodes: 46	accuracy: 0.87	accuracy: 0.865	accuracy: 0.868	accuracy: 0.861	accuracy: 0.859
Nodes: 47	accuracy: 0.855	accuracy: 0.864	accuracy: 0.859	accuracy: 0.862	accuracy: 0.863
Nodes: 48	accuracy: 0.868	accuracy: 0.873	accuracy: 0.856	accuracy: 0.858	accuracy: 0.846
Nodes: 49	accuracy: 0.846	accuracy: 0.853	accuracy: 0.868	accuracy: 0.859	accuracy: 0.859
Nodes: 50	accuracy: 0.866	accuracy: 0.847	accuracy: 0.864	accuracy: 0.864	accuracy: 0.863
Nodes: 51	accuracy: 0.857	accuracy: 0.857	accuracy: 0.866	accuracy: 0.858	accuracy: 0.853
Nodes: 52	accuracy: 0.86	accuracy: 0.863	accuracy: 0.859	accuracy: 0.857	accuracy: 0.854
Nodes: 53	accuracy: 0.867	accuracy: 0.847	accuracy: 0.87	accuracy: 0.859	accuracy: 0.879
Nodes: 54	accuracy: 0.867	accuracy: 0.872	accuracy: 0.869	accuracy: 0.874	accuracy: 0.863

Based on accuracy, we see that the optimal number of nodes varied, as expected. Our initial plan was to take the average of the best node count from each run, however the average of these would be 51 nodes in the hidden layer, which performed poorly across the board, and therefore would not be a good cross validation selection.

Instead, we calculated the average accuracy for each number of nodes across all folds. Figure 3.2.5-4 shows these results.

Figure 3.2.5-3. CV - Average Node Accuracies

	Run					
Nodes	**1**	**2**	**3**	**4**	**5**	**AVG**
42	0.841	0.863	0.861	0.857	0.848	0.854
43	0.858	0.851	0.854	0.853	0.853	0.854
44	0.862	0.859	0.867	0.865	0.859	0.862
45	0.856	0.864	0.854	0.854	0.853	0.856
46	0.870	0.865	0.868	0.861	0.859	0.865
47	0.855	0.864	0.859	0.862	0.863	0.861
48	0.868	0.873	0.856	0.858	0.846	0.860
49	0.846	0.853	0.868	0.859	0.859	0.857
50	0.866	0.847	0.864	0.864	0.863	0.861
51	0.857	0.857	0.866	0.858	0.853	0.858
52	0.860	0.863	0.859	0.857	0.854	0.859
53	0.867	0.847	0.870	0.859	0.879	0.864
54	0.867	0.872	0.869	0.874	0.863	0.869

We see that the number of hidden layer nodes that achieved the highest average accuracy across all folds was 54 nodes, equal to the number of classes (users). We considered going further and trying even larger node counts for the hidden layer, but opted against it since our dataset is so small.

Thus, we selected 54 nodes as our hidden layer size.

Multi-Class Network Performance against Test Set

We train one final model on the full training set, using one hidden layer of 54 nodes, and other parameters as specified above. The overall results are:

- Overall accuracy: 0.874
- Average FRR: 0.127
- Average FAR: 0.002

The Confusion Matrix is shown in Figure 3.2.5-4, and the individual FRR and FAR results for each user are shown in Figure 3.2.5-5.

Figure 3.2.5-4. Confusion Matrix (Multi-Class Neural Network)

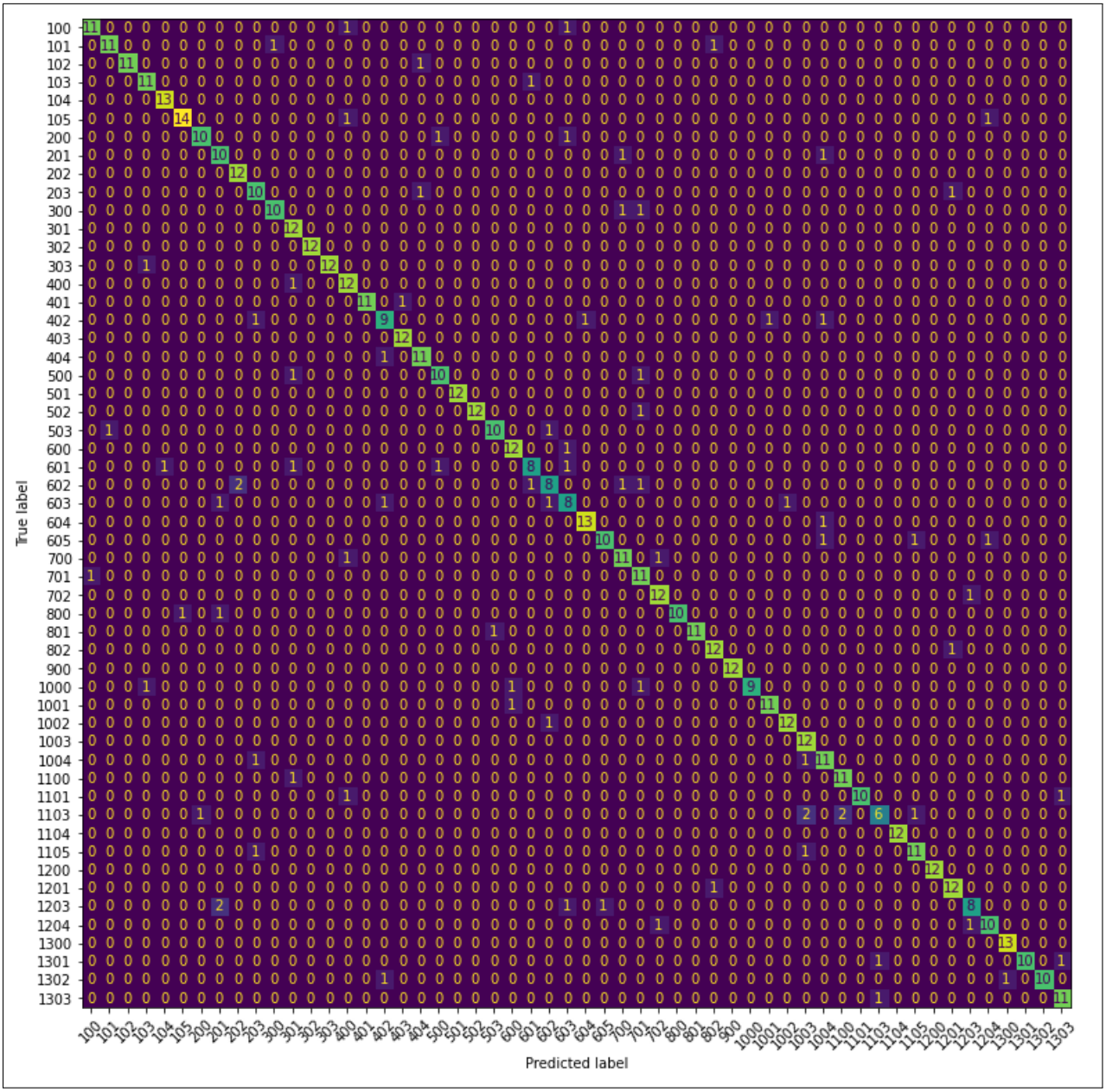


Figure 3.2.5-5. FRR and FAR of Individual Users (Multi-Class Neural Network)

User 100 :	FRR:	0.1538	FAR:	0.0015	User 604 :	FRR:	0.0714	FAR:	0.0015
User 101 :	FRR:	0.1538	FAR:	0.0015	User 605 :	FRR:	0.2308	FAR:	0.0015
User 102 :	FRR:	0.0833	FAR:	0.0000	User 700 :	FRR:	0.1538	FAR:	0.0046
User 103 :	FRR:	0.0833	FAR:	0.0030	User 701 :	FRR:	0.0833	FAR:	0.0076
User 104 :	FRR:	0.0000	FAR:	0.0015	User 702 :	FRR:	0.0769	FAR:	0.0030
User 105 :	FRR:	0.1250	FAR:	0.0015	User 800 :	FRR:	0.1667	FAR:	0.0000
User 200 :	FRR:	0.1667	FAR:	0.0015	User 801 :	FRR:	0.0833	FAR:	0.0000
User 201 :	FRR:	0.1667	FAR:	0.0061	User 802 :	FRR:	0.0769	FAR:	0.0030
User 202 :	FRR:	0.0000	FAR:	0.0030	User 900 :	FRR:	0.0000	FAR:	0.0000
User 203 :	FRR:	0.1667	FAR:	0.0045	User 1000 :	FRR:	0.2500	FAR:	0.0000
User 300 :	FRR:	0.1667	FAR:	0.0015	User 1001 :	FRR:	0.0833	FAR:	0.0015
User 301 :	FRR:	0.0000	FAR:	0.0061	User 1002 :	FRR:	0.0769	FAR:	0.0015
User 302 :	FRR:	0.0000	FAR:	0.0000	User 1003 :	FRR:	0.0000	FAR:	0.0061
User 303 :	FRR:	0.0769	FAR:	0.0000	User 1004 :	FRR:	0.1538	FAR:	0.0061
User 400 :	FRR:	0.0769	FAR:	0.0061	User 1100 :	FRR:	0.0833	FAR:	0.0030
User 401 :	FRR:	0.0833	FAR:	0.0000	User 1101 :	FRR:	0.1667	FAR:	0.0000
User 402 :	FRR:	0.3077	FAR:	0.0046	User 1103 :	FRR:	0.5000	FAR:	0.0030
User 403 :	FRR:	0.0000	FAR:	0.0015	User 1104 :	FRR:	0.0000	FAR:	0.0000
User 404 :	FRR:	0.0833	FAR:	0.0030	User 1105 :	FRR:	0.1538	FAR:	0.0030
User 500 :	FRR:	0.1667	FAR:	0.0030	User 1200 :	FRR:	0.0000	FAR:	0.0000
User 501 :	FRR:	0.0000	FAR:	0.0000	User 1201 :	FRR:	0.0769	FAR:	0.0030
User 502 :	FRR:	0.0769	FAR:	0.0000	User 1203 :	FRR:	0.3333	FAR:	0.0030
User 503 :	FRR:	0.1667	FAR:	0.0015	User 1204 :	FRR:	0.1667	FAR:	0.0030
User 600 :	FRR:	0.0769	FAR:	0.0030	User 1300 :	FRR:	0.0000	FAR:	0.0015
User 601 :	FRR:	0.3333	FAR:	0.0030	User 1301 :	FRR:	0.1667	FAR:	0.0000
User 602 :	FRR:	0.3846	FAR:	0.0046	User 1302 :	FRR:	0.1667	FAR:	0.0000
User 603 :	FRR:	0.3333	FAR:	0.0076	User 1303 :	FRR:	0.0833	FAR:	0.0030

Overall, the model has decent accuracy, performing slightly better than our very first naive run, pre-feature selection. Also the model has good overall FAR, meaning that most attacks will be correctly flagged. However, the average FRR metric is high. So, almost 13% of the time, a user will be denied access to their own device. Now, some organizations might feel this is an acceptable price to pay for high security if all that the user has to do is re-enter their password, but 13% is much worse than the industry standard convention of around 1%. Furthermore, if we look deeper into the FAR and FRR metrics for each *individual user*, we see that many users have an unacceptably high FRR, with several users being rejected from logging into their device more than 30% of the time. For example, user ID 602 has a false rejection rate of almost 40%! Such results are unacceptable. Thus, this model should not be used as a basis for which to deny users access to their devices.

To that end, we now train individual neural networks for each individual user, to see if we can mitigate this issue and get the FRR for individual users down to a more reasonable level.

Binary Classification Network Cross Validation

Recall that when training a binary classification model for each user, there are only two (2) nodes in the output layer of the network: one for the user, one for all other users. For the multi-class model, we tried all node sizes between the input layer size and the output layer size, so we considered doing the same here, but decided to still include values up to 54, since there are still that many unique users in the dataset, and it could be helpful to have a node for each.

Additionally, we tune the 'alpha' parameter (which is a regularization parameter) and the learning_rate_init parameter, because in these individiual binary classification models, they both seem to have an effect on the FRR, which we are trying to minimize. Neither had any effect on the multi-class neural network, thus we omitted them as parameters to tune and just used the default.

Note that we actually had to *increase* the learning rate for this user (and for many other users as well) in order to get better performance. Perhaps starting with such a small learning rate was too easily getting trapped in local minima, which is why starting out with a larger initial learning rate and then adaptively reducing it results in good performance for these individual user models.

Figure 3.2.5-6 shows the selected parameters for all individual models.

Figure 3.2.5-6. Selected Parameters for User Models (Binary-Classification Neural Network)

User 100 :	alpha: 1	learning_rate_init: 0.01		User 604 :	alpha: 0.01	learning_rate_init: 0.001	
User 101 :	alpha: 0.01	learning_rate_init: 0.001		User 605 :	alpha: 0.01	learning_rate_init: 0.001	
User 102 :	alpha: 1	learning_rate_init: 0.01		User 700 :	alpha: 1	learning_rate_init: 0.01	
User 103 :	alpha: 1	learning_rate_init: 0.01		User 701 :	alpha: 1	learning_rate_init: 0.01	
User 104 :	alpha: 1	learning_rate_init: 0.01		User 702 :	alpha: 1	learning_rate_init: 0.01	
User 105 :	alpha: 1	learning_rate_init: 0.01		User 800 :	alpha: 0.01	learning_rate_init: 0.001	
User 200 :	alpha: 1	learning_rate_init: 0.01		User 801 :	alpha: 1	learning_rate_init: 0.01	
User 201 :	alpha: 1	learning_rate_init: 0.01		User 802 :	alpha: 0.01	learning_rate_init: 0.001	
User 202 :	alpha: 0.01	learning_rate_init: 0.001		User 900 :	alpha: 1	learning_rate_init: 0.01	
User 203 :	alpha: 0.01	learning_rate_init: 0.001		User 1000 :	alpha: 0.0001	learning_rate_init: 0.01	
User 300 :	alpha: 1	learning_rate_init: 0.01		User 1001 :	alpha: 1	learning_rate_init: 0.01	
User 301 :	alpha: 1	learning_rate_init: 0.01		User 1002 :	alpha: 0.01	learning_rate_init: 0.001	
User 302 :	alpha: 0.01	learning_rate_init: 0.001		User 1003 :	alpha: 0.01	learning_rate_init: 0.001	
User 303 :	alpha: 0.01	learning_rate_init: 0.001		User 1004 :	alpha: 1	learning_rate_init: 0.01	
User 400 :	alpha: 1	learning_rate_init: 0.01		User 1100 :	alpha: 0.01	learning_rate_init: 0.001	
User 401 :	alpha: 1	learning_rate_init: 0.01		User 1101 :	alpha: 1	learning_rate_init: 0.01	
User 402 :	alpha: 1	learning_rate_init: 0.01		User 1103 :	alpha: 0.001	learning_rate_init: 0.01	
User 403 :	alpha: 0.01	learning_rate_init: 0.001		User 1104 :	alpha: 0.01	learning_rate_init: 0.001	
User 404 :	alpha: 0.01	learning_rate_init: 0.001		User 1105 :	alpha: 0.01	learning_rate_init: 0.001	
User 500 :	alpha: 1	learning_rate_init: 0.01		User 1200 :	alpha: 0.01	learning_rate_init: 0.001	
User 501 :	alpha: 0.01	learning_rate_init: 0.001		User 1201 :	alpha: 1	learning_rate_init: 0.01	
User 502 :	alpha: 10	learning_rate_init: 0.01		User 1203 :	alpha: 1	learning_rate_init: 0.01	
User 503 :	alpha: 0.01	learning_rate_init: 0.001		User 1204 :	alpha: 1	learning_rate_init: 0.01	
User 600 :	alpha: 1	learning_rate_init: 0.01		User 1300 :	alpha: 1	learning_rate_init: 0.01	
User 601 :	alpha: 1	learning_rate_init: 0.01		User 1301 :	alpha: 0.01	learning_rate_init: 0.001	
User 602 :	alpha: 0.01	learning_rate_init: 0.001		User 1302 :	alpha: 1	learning_rate_init: 0.01	
User 603 :	alpha: 1	learning_rate_init: 0.01		User 1303 :	alpha: 0.01	learning_rate_init: 0.001	

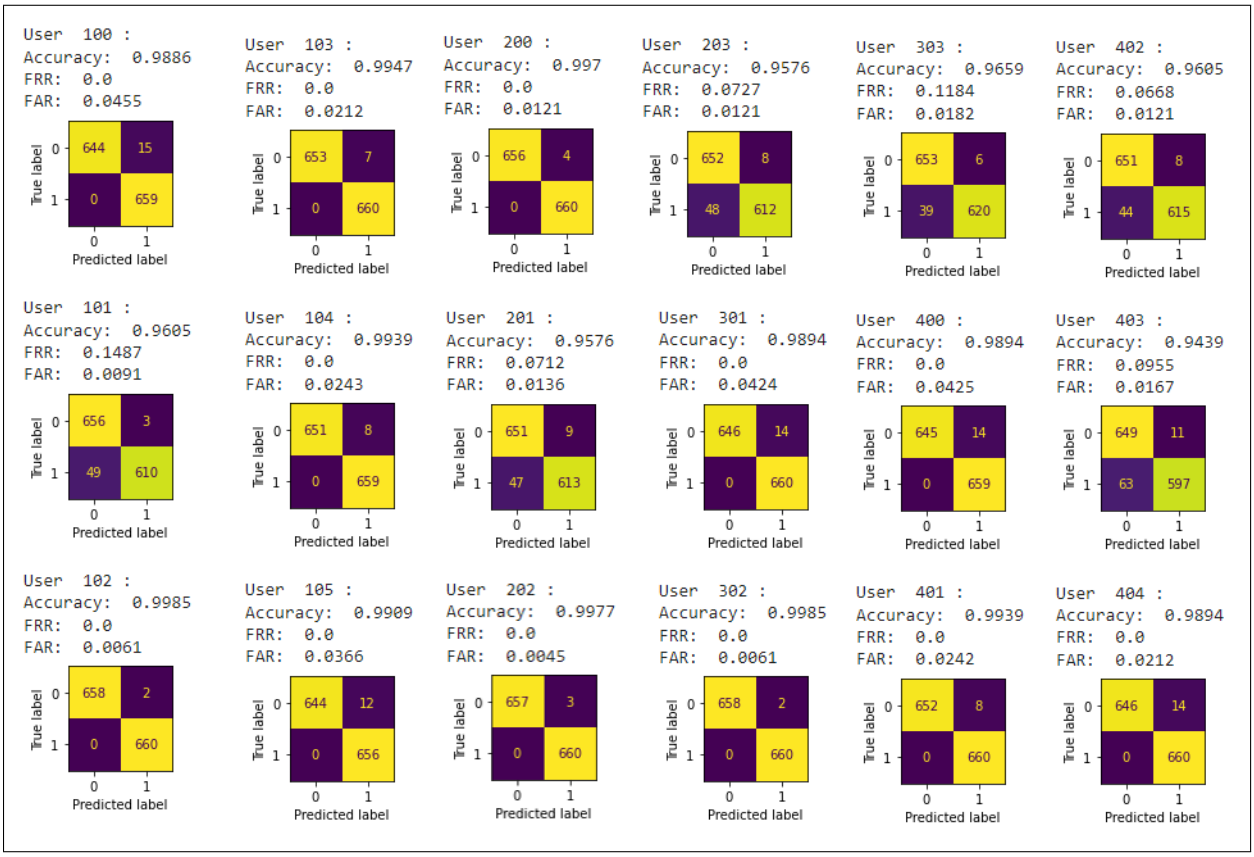
Binary Classification Network Performance Against Test Set

We train one final model (for each user) using these tuned parameters on the full training set. The overall results averaged across all users are:

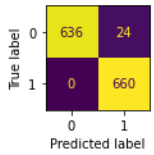
- Average accuracy: 0.981
- Average FRR: .029
- Average FAR: .019

Figure 3.2.5-7 shows the confusion matrices of the final trained model for all users when evaluated against the test set, and Figure 3.2.5-8 shows their detailed metrics.

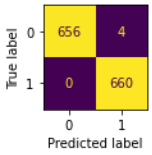
Figure 3.2.5-7. Individual Confusion Matrices (Binary-Classification Neural Network)



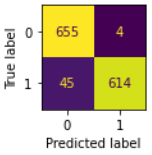
User 500 :
Accuracy: 0.9818
FRR: 0.0
FAR: 0.0727



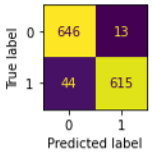
User 503 :
Accuracy: 0.997
FRR: 0.0
FAR: 0.0061



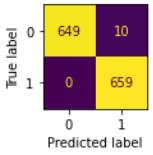
User 602 :
Accuracy: 0.9628
FRR: 0.0683
FAR: 0.0061



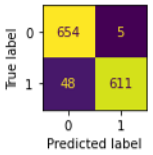
User 605 :
Accuracy: 0.9568
FRR: 0.1335
FAR: 0.0395



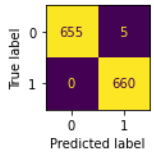
User 702 :
Accuracy: 0.9924
FRR: 0.0
FAR: 0.0152



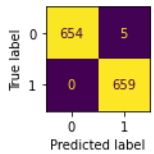
User 802 :
Accuracy: 0.9598
FRR: 0.0728
FAR: 0.0076



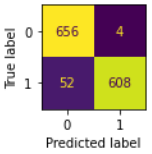
User 501 :
Accuracy: 0.9962
FRR: 0.0
FAR: 0.0152



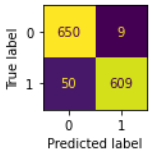
User 600 :
Accuracy: 0.9962
FRR: 0.0
FAR: 0.0152



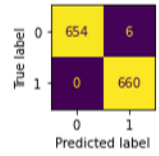
User 603 :
Accuracy: 0.9576
FRR: 0.0788
FAR: 0.0061



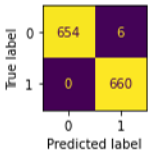
User 700 :
Accuracy: 0.9552
FRR: 0.0759
FAR: 0.0137



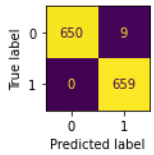
User 800 :
Accuracy: 0.9955
FRR: 0.0
FAR: 0.0182



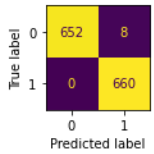
User 900 :
Accuracy: 0.9955
FRR: 0.0
FAR: 0.0091



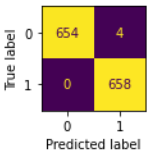
User 502 :
Accuracy: 0.9932
FRR: 0.0
FAR: 0.0273



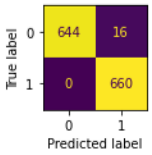
User 601 :
Accuracy: 0.9939
FRR: 0.0
FAR: 0.0121



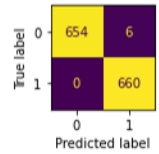
User 604 :
Accuracy: 0.997
FRR: 0.0
FAR: 0.0061



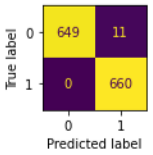
User 701 :
Accuracy: 0.9879
FRR: 0.0
FAR: 0.0485



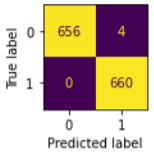
User 801 :
Accuracy: 0.9955
FRR: 0.0
FAR: 0.0182



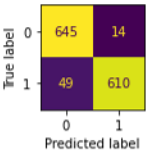
User 1000 :
Accuracy: 0.9917
FRR: 0.0
FAR: 0.0167



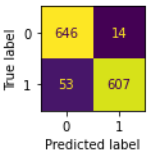
User 1001 :
Accuracy: 0.997
FRR: 0.0
FAR: 0.0061



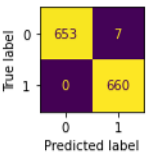
User 1004 :
Accuracy: 0.9522
FRR: 0.0744
FAR: 0.0212



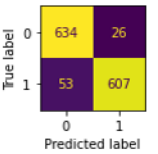
User 1103 :
Accuracy: 0.9492
FRR: 0.0803
FAR: 0.0212



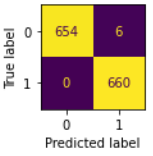
User 1200 :
Accuracy: 0.9947
FRR: 0.0
FAR: 0.0212



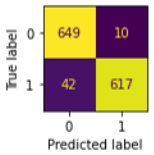
User 1204 :
Accuracy: 0.9402
FRR: 0.0803
FAR: 0.0394



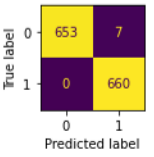
User 1302 :
Accuracy: 0.9955
FRR: 0.0
FAR: 0.0091



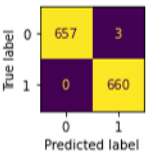
User 1002 :
Accuracy: 0.9605
FRR: 0.0637
FAR: 0.0152



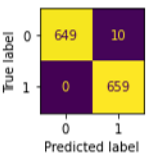
User 1100 :
Accuracy: 0.9947
FRR: 0.0
FAR: 0.0106



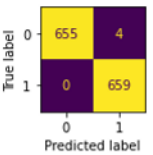
User 1104 :
Accuracy: 0.9977
FRR: 0.0
FAR: 0.0091



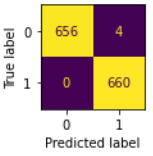
User 1201 :
Accuracy: 0.9924
FRR: 0.0
FAR: 0.0303



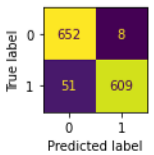
User 1300 :
Accuracy: 0.997
FRR: 0.0
FAR: 0.0061



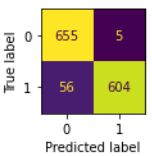
User 1303 :
Accuracy: 0.997
FRR: 0.0
FAR: 0.0061



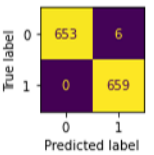
User 1003 :
Accuracy: 0.9553
FRR: 0.0773
FAR: 0.0121



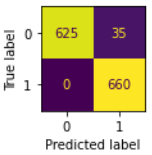
User 1101 :
Accuracy: 0.9538
FRR: 0.0848
FAR: 0.0076



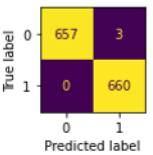
User 1105 :
Accuracy: 0.9954
FRR: 0.0
FAR: 0.0182



User 1203 :
Accuracy: 0.9735
FRR: 0.0
FAR: 0.053



User 1301 :
Accuracy: 0.9977
FRR: 0.0
FAR: 0.0045



User 301 :
Accuracy: 0.9477
FRR: 0.0894
FAR: 0.0152

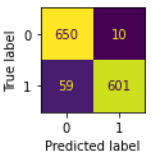


Figure 3.2.5-8. Individual Confusion Matrices (Binary-Classification Neural Network)

User 100 :	Accuracy:	0.9886	FRR:	0.0	FAR:	0.0455	User 604 :	Accuracy:	0.9552	FRR:	0.0759	FAR:	0.0137
User 101 :	Accuracy:	0.9605	FRR:	0.0	FAR:	0.0212	User 605 :	Accuracy:	0.9955	FRR:	0.0	FAR:	0.0182
User 102 :	Accuracy:	0.9985	FRR:	0.0	FAR:	0.0121	User 700 :	Accuracy:	0.9955	FRR:	0.0	FAR:	0.0091
User 103 :	Accuracy:	0.9947	FRR:	0.0727	FAR:	0.0121	User 701 :	Accuracy:	0.9932	FRR:	0.0	FAR:	0.0273
User 104 :	Accuracy:	0.9939	FRR:	0.1184	FAR:	0.0182	User 702 :	Accuracy:	0.9939	FRR:	0.0	FAR:	0.0121
User 105 :	Accuracy:	0.9909	FRR:	0.0668	FAR:	0.0121	User 800 :	Accuracy:	0.997	FRR:	0.0	FAR:	0.0061
User 200 :	Accuracy:	0.997	FRR:	0.1487	FAR:	0.0091	User 801 :	Accuracy:	0.9879	FRR:	0.0	FAR:	0.0485
User 201 :	Accuracy:	0.9576	FRR:	0.0	FAR:	0.0243	User 802 :	Accuracy:	0.9955	FRR:	0.0	FAR:	0.0182
User 202 :	Accuracy:	0.9977	FRR:	0.0712	FAR:	0.0136	User 900 :	Accuracy:	0.9917	FRR:	0.0	FAR:	0.0167
User 203 :	Accuracy:	0.9576	FRR:	0.0	FAR:	0.0424	User 1000 :	Accuracy:	0.997	FRR:	0.0	FAR:	0.0061
User 300 :	Accuracy:	0.9894	FRR:	0.0	FAR:	0.0425	User 1001 :	Accuracy:	0.9522	FRR:	0.0744	FAR:	0.0212
User 301 :	Accuracy:	0.9985	FRR:	0.0955	FAR:	0.0167	User 1002 :	Accuracy:	0.9492	FRR:	0.0803	FAR:	0.0212
User 302 :	Accuracy:	0.9659	FRR:	0.0	FAR:	0.0061	User 1003 :	Accuracy:	0.9947	FRR:	0.0	FAR:	0.0212
User 303 :	Accuracy:	0.9894	FRR:	0.0	FAR:	0.0366	User 1004 :	Accuracy:	0.9402	FRR:	0.0803	FAR:	0.0394
User 400 :	Accuracy:	0.9939	FRR:	0.0	FAR:	0.0045	User 1100 :	Accuracy:	0.9955	FRR:	0.0	FAR:	0.0091
User 401 :	Accuracy:	0.9605	FRR:	0.0	FAR:	0.0061	User 1101 :	Accuracy:	0.9605	FRR:	0.0637	FAR:	0.0152
User 402 :	Accuracy:	0.9439	FRR:	0.0	FAR:	0.0242	User 1103 :	Accuracy:	0.9947	FRR:	0.0	FAR:	0.0106
User 403 :	Accuracy:	0.9894	FRR:	0.0	FAR:	0.0212	User 1104 :	Accuracy:	0.9977	FRR:	0.0	FAR:	0.0091
User 404 :	Accuracy:	0.9818	FRR:	0.0	FAR:	0.0727	User 1105 :	Accuracy:	0.9924	FRR:	0.0	FAR:	0.0303
User 500 :	Accuracy:	0.997	FRR:	0.0	FAR:	0.0061	User 1200 :	Accuracy:	0.997	FRR:	0.0	FAR:	0.0061
User 501 :	Accuracy:	0.9628	FRR:	0.0683	FAR:	0.0061	User 1201 :	Accuracy:	0.997	FRR:	0.0	FAR:	0.0061
User 502 :	Accuracy:	0.9568	FRR:	0.1335	FAR:	0.0395	User 1203 :	Accuracy:	0.9553	FRR:	0.0773	FAR:	0.0121
User 503 :	Accuracy:	0.9924	FRR:	0.0	FAR:	0.0152	User 1204 :	Accuracy:	0.9538	FRR:	0.0848	FAR:	0.0076
User 600 :	Accuracy:	0.9598	FRR:	0.0728	FAR:	0.0076	User 1300 :	Accuracy:	0.9954	FRR:	0.0	FAR:	0.0182
User 601 :	Accuracy:	0.9962	FRR:	0.0	FAR:	0.0152	User 1301 :	Accuracy:	0.9735	FRR:	0.0	FAR:	0.053
User 602 :	Accuracy:	0.9962	FRR:	0.0	FAR:	0.0152	User 1302 :	Accuracy:	0.9977	FRR:	0.0	FAR:	0.0045
User 603 :	Accuracy:	0.9576	FRR:	0.0788	FAR:	0.0061	User 1303 :	Accuracy:	0.9477	FRR:	0.0894	FAR:	0.0152

Overall, this system of models has a much higher overall accuracy than that of the multi-class neural network, and the overall FRR is now within acceptable limits as well. Most users experience zero (0) false rejections, some experience rejections in the single-digit percentages (not terrible), and only a couple have more unacceptable rejection rates in the low teens. However, this system of models also has a significantly worse FAR than the multi-class model above, although still likely within the realm of acceptability since this is a secondary security measure.

Given this, a more robust security solution would be to utilize both the binary classification models as well as the multi-class model in a joint fashion. For example, perhaps each user's device could have their particular trained binary classification model loaded onto their device, but no one else's. (This is good operational security practice.) If the user's device detected an attempted intrusion and denied access, it could transmit the data point back to a central server of the organization, which could then run the data point through the multi-class model as well as through all the other users' individual binary classification models to determine if it was likely an actual insider or an outside attacker. Any users flagged as insiders (potentially beyond some threshold) could be further investigated, perhaps by checking their badge-in/badge-out records or investigating alibis.

3.2.6 Decision Trees and Random Forest

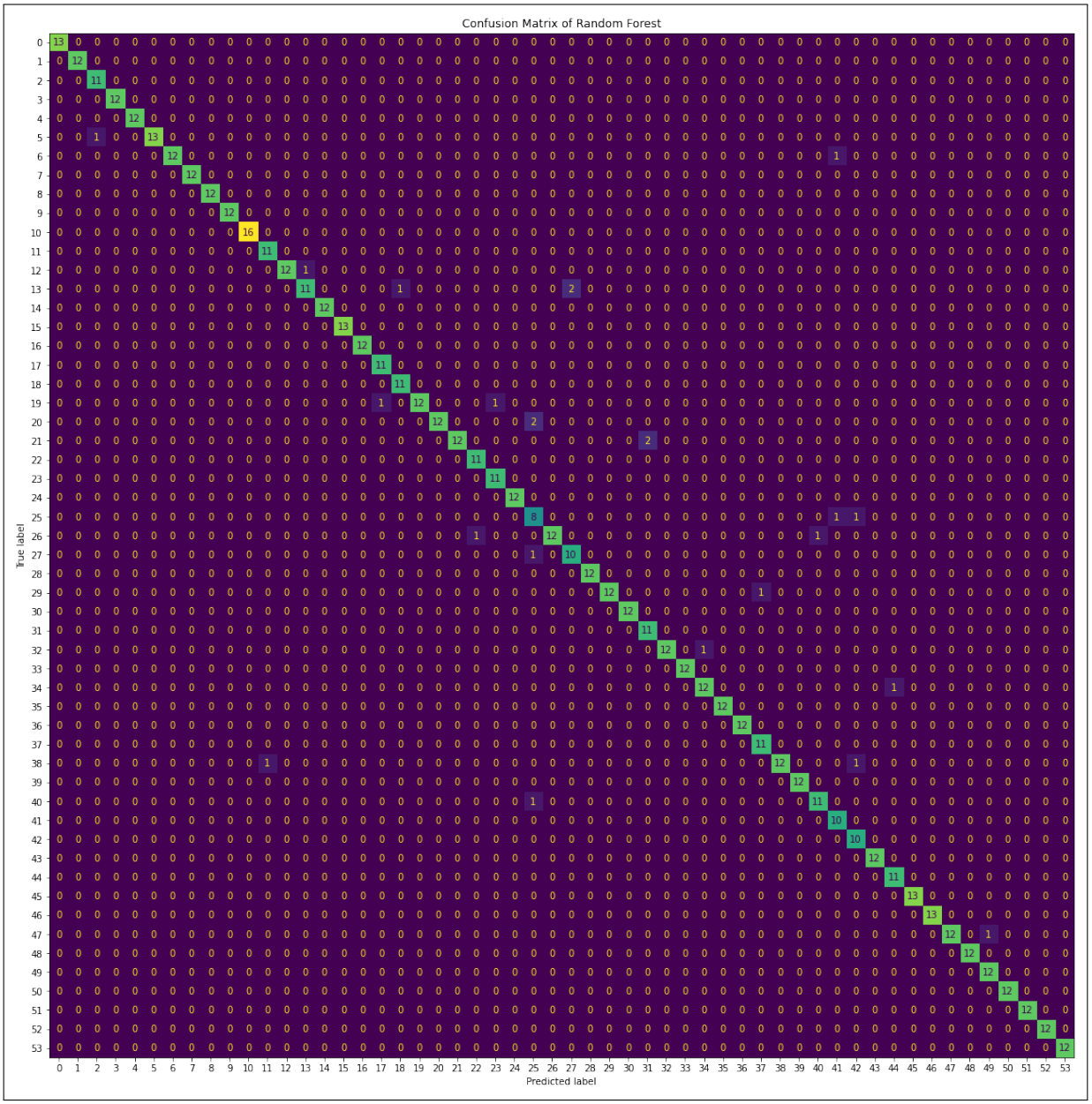
While the train time complexity for a decision tree ($O(ndlog(n))$ where d is the number of variables) is reasonable, the random forest train time complexity is a bit more complicated, as it involves both n and $log(n)$, but instead of d, we have the number of variables that are chosen to be a part of any given tree, m. At this point, for a random forest of one tree, we reach time complexity $O(nmlog(n))$, where $m < d$, $\forall m$. The benefit of the random forest over the decision tree, which must be iterated over with many choices of depth, is that the decision tree is often artificially highly accurate, and does not generalize well. The random forest is capable of generalizing well by performing this computation many times, each time choosing a different m-size subset of d. This will be the number of iterations i, which leads to a time complexity of $O(nmilog(n))$. The trade-off lies in the need for highly important selectivity of pruning for developing an accurate decision tree, while assessing over fitment of the model. For the purposes of implementing the a system like this, the choice of random forests was easy.

Random Forest returned near-perfect scores on raw data, which raised red flags. Returning to the the model, we performed a repeated stratified K fold cross-validation to ensure results, and after comparing 50 outcomes, we settle on a 96.4% accuracy. This was performed as a multi-class problem, with all but 1 user presenting a FAR of more than 8.3%. In practice, this one user would likely be able to request some other kind of backup authentication if the detection software continues to prove disruptive to their productivity.

The repeated cross-validation ensures that the model is not overfit, and that its average accuracy score is reported correctly.

The Random Forest The random forest is a bagging algorithm that uses, as its base, the same process used in the building of a decision tree. The random forest adds to this model by selecting some subset of features from the data and building a smaller decision tree. These less-informed decision trees have been built with random sampling that we expect to, on average, produce the correct decision. As such, a record of the classification of each vote is taken from each smaller decision tree, and a majority vote of the intended label is designated as true. The bootstrap resampling produces high variance, which is mitigated by pooling in the majority vote process of selecting the class.

Figure 3.2.6-1. Confusion Matrix (Multi-Classification Random Forest)



3.3 Post-Authentication

For the use case of detecting an unauthorized user on an already-unlocked device, we must simulate the effect of continuously typed data, beginning AFTER initial authentication has been granted (i.e., AFTER the login screen). In a real-world scenario, the assumption is that after the user logs into their device, they would begin using it, and would begin typing. Keytap data would be collected for each key as it was tapped, and this data would be streamed into a change detection model.

In order to simulate this effect, since the MOBIKEY dataset only contains data points for whole passwords, we extract out the features for each individual key character, and combine the features that pertain to a given character as one data point.

In a real world scenario, a more robust training / data collection activity could be performed so more data points could be gathered , and a model and/or a distribution could then be obtained for each character. Then, during use, after model training, as a user taps characters on the touchscreen keyboard, each one would be fed into its respective model (i.e., the model for that character), and deviation from the model could be determined.

Since we do not have that many data points, we choose to focus on the letter 'k' as a proof of concept, since this letter occurs the most frequently (5 occurrences) in our dataset, taking into account all passwords:

- Easy: kicsikutyatarka
- Logical Strong: Kktsf2!2014
- Strong: .tie5Roanl

A production-quality system would likely take into account multiple letters and multiple pairs of letters based on their frequency in the target language. (Refer to Future Work discussion below.)

Feature Selection

Note that the 2nd order features are not relevant to the use case of post-authentication, as the 2nd order features were features for password entries as a whole, not for individual letters. Furthermore, for this proof-of-concept, we opt to focus on only those first order features that pertain to a single key character, not pairs of characters. That is, we reduce our feature set to only 'Hold time (HT)', 'Pressure (P)', and 'Finger Area (FA)' for the key character 'k'. The 'Down-Down (DD)' and 'Up-Down' features are excluded.

Notice that the letter 'k' corresponds to keytaps 1, 6, and 14 in the 'Easy' password, and to keytaps 2 and 3 in the 'Logical Strong' password. (Note: the first keytap in the Logical Strong password is actually the 'shift' key.) Thus, we create individual data points for the letter 'k' by extracting the following groups of features from the dataset:

- Easy:
  - holdtime1, pressure1, fingerarea1
  - holdtime6, pressure6, fingerarea6
  - holdtime14, pressure14, fingerarea14
- Logical Strong:
  - holdtime2, pressure2, fingerarea2
  - holdtime3, pressure3, fingerarea3

Thus, our feature set has been reduced to 3 dimensions: 'Hold time (HT)', 'Pressure (P)', and 'Finger Area (FA)'. No further feature selection is necessary.

After collecting the data in this way, we then split and standardize the data as detailed in section 3.1.3 Data Preparation (Splitting and Standardization) above.

Simulation

In order to simulate an authenticated user using their device for some amount of time, after which an attacker obtains access to the already unlocked device and begins using it, we randomly select two (2) users: one to be the authenticated user, and one to be the attacker. We then randomly select 100 data points for each of those two users and concatenate them such that the first 100 data points are those of the authenticated user, and the subsequent 100 data points are those of the attacker. The data points will then be ingested sequentially into our model as a time series, wherein time $t = 1$ corresponds to the first data point, time $t = 2$ corresponds to the 2nd data point, and so on. Our goal will be to detect the change and identify that an attack has occurred. In a real world scenario, an attack detection could trigger any number of actions, such as forcing the device to lock.

3.3.1 GLR (Change Detection)

Our initial plan was to utilize CUSUM as our change detection method, wherein for each time t, we would update the CUSUM statistic. Recall the statistic:

$$W_{t+1} = max\left\{0,\ W_t + log\left(\frac{f_1(X)}{f_0(X)}\right)\right\}$$

where:

- $W_0 = 0$
- $f_0(X)$ is the distribution of the data points *before* the change
- $f_1(X)$ is the distribution of the data points *after* the change

Knowing both the pre- and post-change distributions makes sense in such use cases as factory equipment, where an organization might already know how equipment behaves once it malfunctions.
However, for our use case, this 'post-change' distribution cannot be known. For us, the distribution *before* the change corresponds to the distribution of the current authenticated user. The *post-change* distribution corresponds to the attacker. In a real-world scenario, we would not know the distribution of the attacker, since we would not know who they are. We would only know the distribution of the currently authenticated user. Therefore, we cannot use CUSUM.

The authors then discovered a paper [2] by Xie et al, (further explanation of which is presented in the slides in [3]) detailing a different method for change detection when the post-change distribution is not known. The method in this paper utilizes the Generalized Likelihood Ratio (GLR) statistic. It essentially looks for a significant deviation in the mean of the data from that of the *pre*-change distribution. The GLR statistic is formed by replacing the unknown mean (μ) with its maximum likelihood estimator. A sliding window is used to look at data points from the current time t back a certain number of timeframes, to just after time k, which the method then claims is the change point if a change is detected. The GLR statistic is monitored for the window $[k + 1, t]$, and the process is deemed to be out of control if the GLR statistic ever exceeds some threshold b during this window.

The GLR statistic is given by the following formula:

$$\bar{x}_{s,t} = \frac{1}{t-k}\sum_{i=k+1}^{t} x_i$$

Given a pre-change distribution with mean μ, the stopping criteria for this method would be when the difference between the two means exceeds some threshold b. That is, when

$$\frac{1}{t-k}\sum_{i=k+1}^{t} x_i - \mu \geq b$$

Note that in the paper, the assumption was made that the mean of the pre-change distribution was zero (0), thus changing this statistic to be simply:

$$\frac{1}{t-k}\sum_{i=k+1}^{t} x_i \geq b$$

For our use case, however, the mean of the pre-change distribution (i.e., the mean of the authenticated user) will not be zero. Since we standardize data from all users together, the mean of each user will not necessarily be zero. Thus, we need to measure the difference between the mean of the authenticated user and that of the actual data before comparing to the threshold b. We can do this, because we can make the assumption that all the users in an organization would be a reasonable representative sample for a population, thus we can identify where each user in our organization falls with respect to the population.

For our use case of post-authentication change detection, armed with the knowledge from the aforementioned paper, we can proceed in the following manner:

1. Learn the distribution of each user using the data from the training set
2. Learn the threshold b for each user, by running simulations of the user being attacked by each other user
3. Perform cross-validation by performing five (5) runs for each attacker attacking the current user
4. Take b to be the average b from all five (5) simulation runs
5. Perform a control experiment where no attack occurs, wherein all data points are from the current user
6. Use this learned threshold b and run the simulation using the test set
7. Construct confusion matrix and evaluate performance of our model

Step 1: Learn the distribution of each user

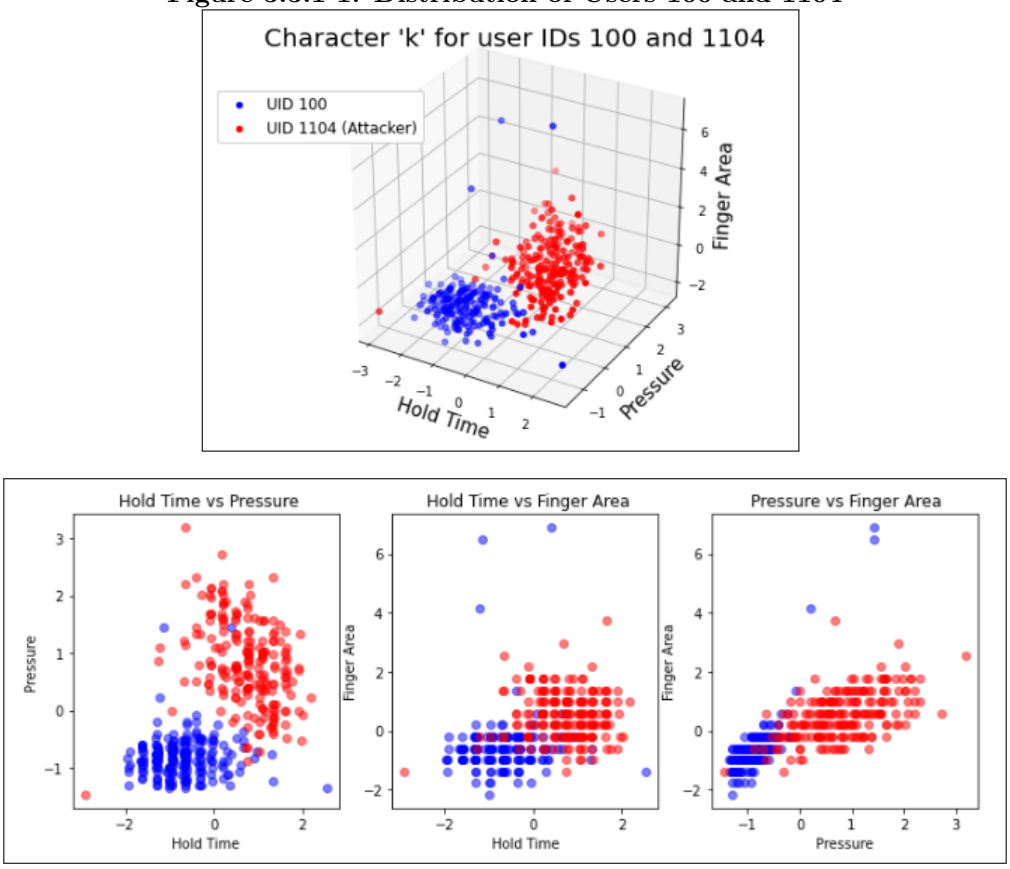
We start by taking the first user of the dataset (user ID 100) to be the current authenticated user. We calculate the mean data point for the current user by taking the mean of each feature (column) and combining them into one mean data point. For user 100, this mean is:

$$\mu_{100} = (-0.62568521,\ 0.42386232,\ 0.15770849)$$

For illustrative purposes, we plot user 100's keytap data against each other user (attacker) to get a feel for how users compare. Recall that our data is three-dimensional, since we have three (3) features. We plot the 3D plot as well as the 2D plots for each pair of features. The current user (user 100) is shown in blue, and attackers are shown in red.

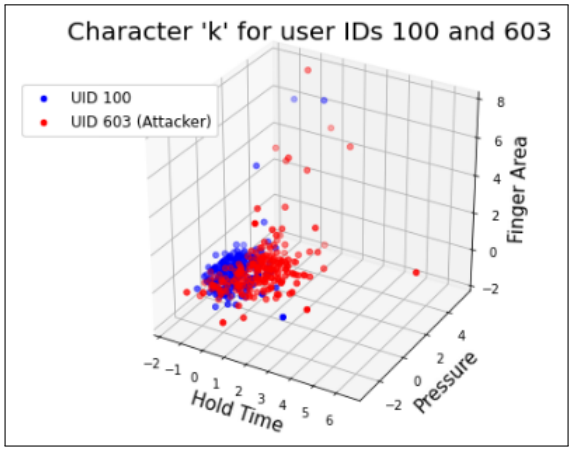
We see that some users are very different from user 100, such as user 1104, making them easy to distinguish.

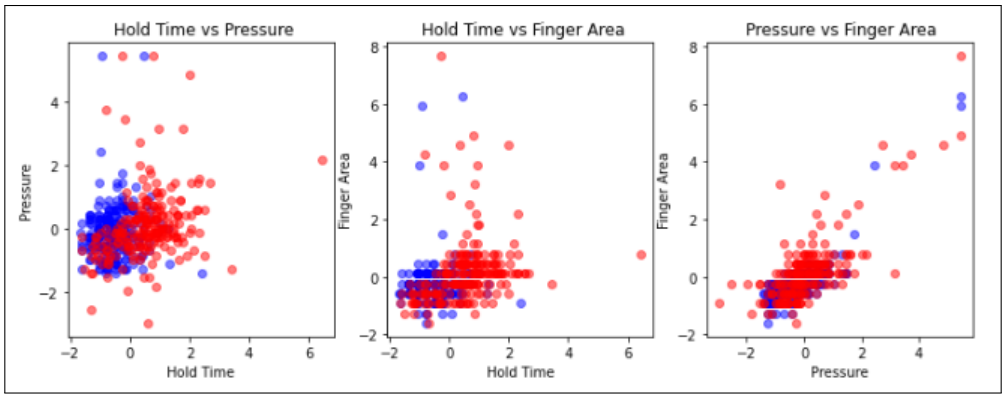
Figure 3.3.1-1. Distribution of Users 100 and 1104



Whereas other users are very similar to user 100, such as user 6, making them more difficult to distinguish.

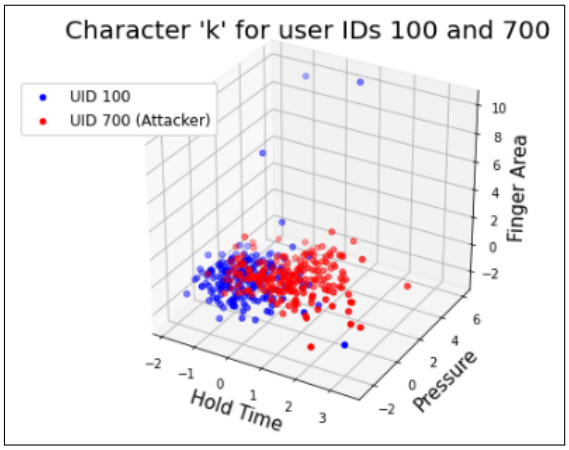
Figure 3.3.1-2. Distribution of Users 100 and 603

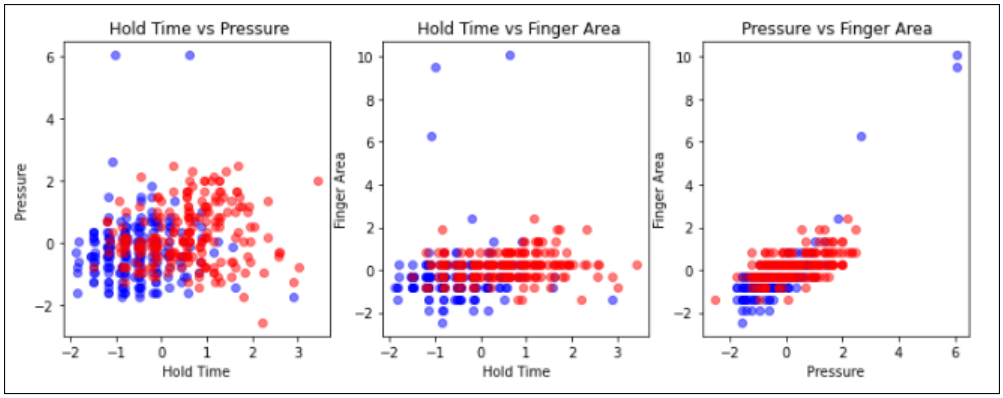




Yet, we notice that most user pairings are somewhere in between:

Figure 3.3.1-3. Distribution of Users 100 and 700





So the question is: Can we detect a shift in the mean once an attacker obtains and begins using the unlocked device?

Step 2: Learn the threshold b for each user

Taking one user to be the authenticated user, we perform one simulation with each other user (as the attacker). That is, since we have 54 users total, one of which we have taken to be the authenticated user, we will run 53 simulations. Each simulation has a different attacker.

We use a sliding window of 50 total. That is, we wait for the first 50 datapoints to occur, and we calculate the GLR statistic:

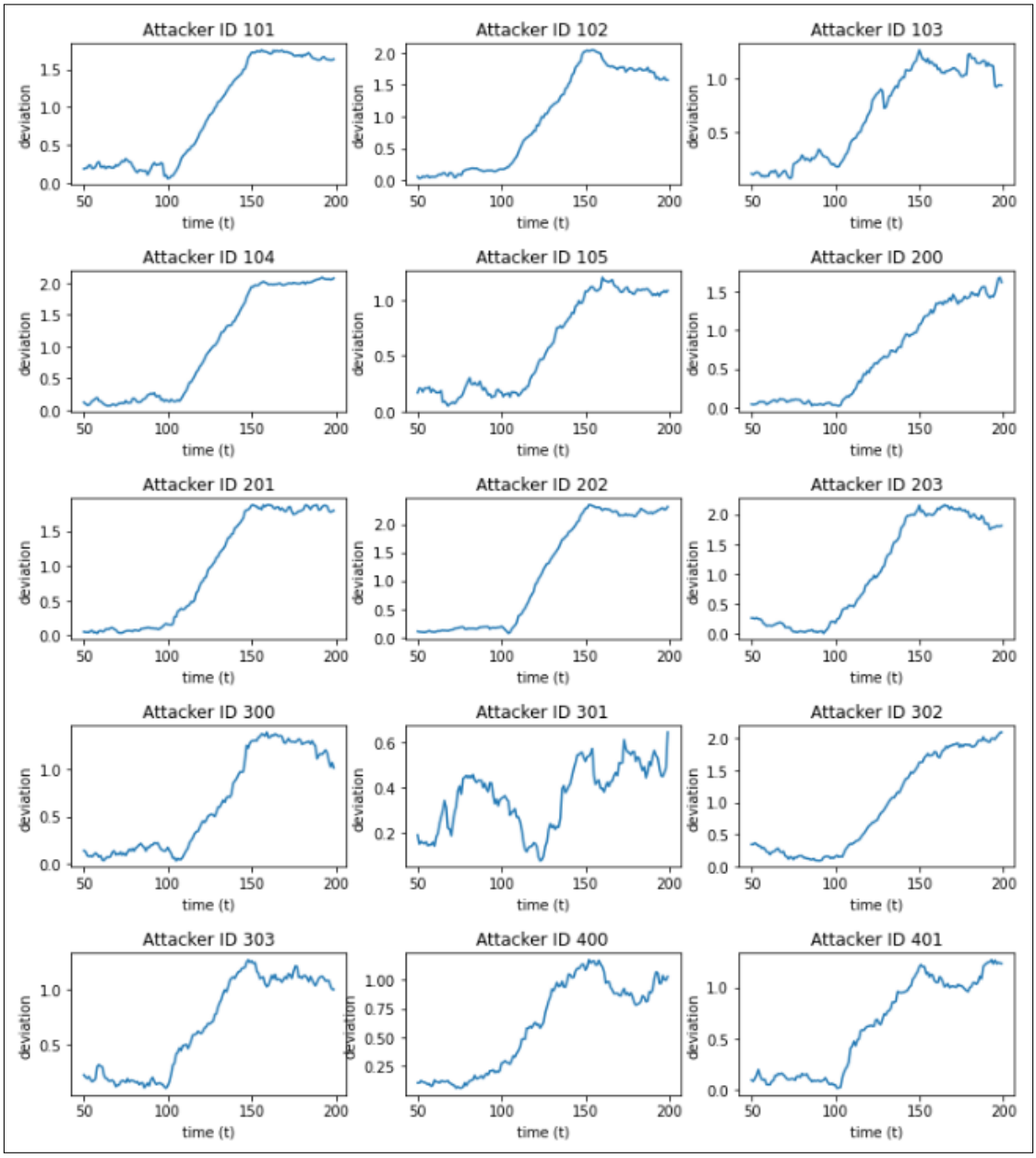
$$\frac{1}{t-k}\sum_{i=k+1}^{t} x_i - \mu$$

We start from $t = 51$, and we take $k = 50$. For our current example of user 100, our formula for the GLR statistic becomes:

$$\frac{1}{t-50}\sum_{i=50+1}^{t} x_i - (-0.62568521,\ 0.42386232,\ 0.15770849)$$

Since our data is three-dimensional, calculate $x_i - \mu$ as the euclidean distance (ℓ_2-norm) between them. We plot the GLR statistic for each 50-data-point window. Since our simulation contains 200 total data points, we have $200 - 50 = 150$ data point windows, one GLR statistic each. Figure 3.3.1-4. GLR Statistic of each Attacker shows the plots of the first 15 attackers. (The remainder are omitted from this report for brevity, yet they were included in the code.)

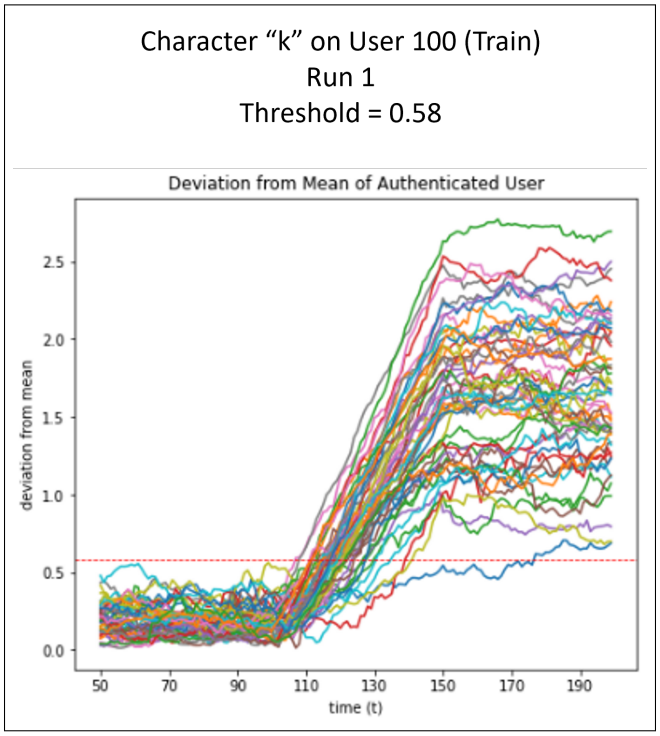
Figure 3.3.1-4. GLR Statistic of each Attacker



Notice that some control charts are much cleaner than others. This is due to the similarity between the user keytap behavior and that of the attacker.

To determine the threshold b, we consider the space of all attackers. To visualize this, we plot the control charts of all attackers together in one. See Figure 3.3.1-5.

Figure 3.3.1-5. GLR Statistics of All Attackers

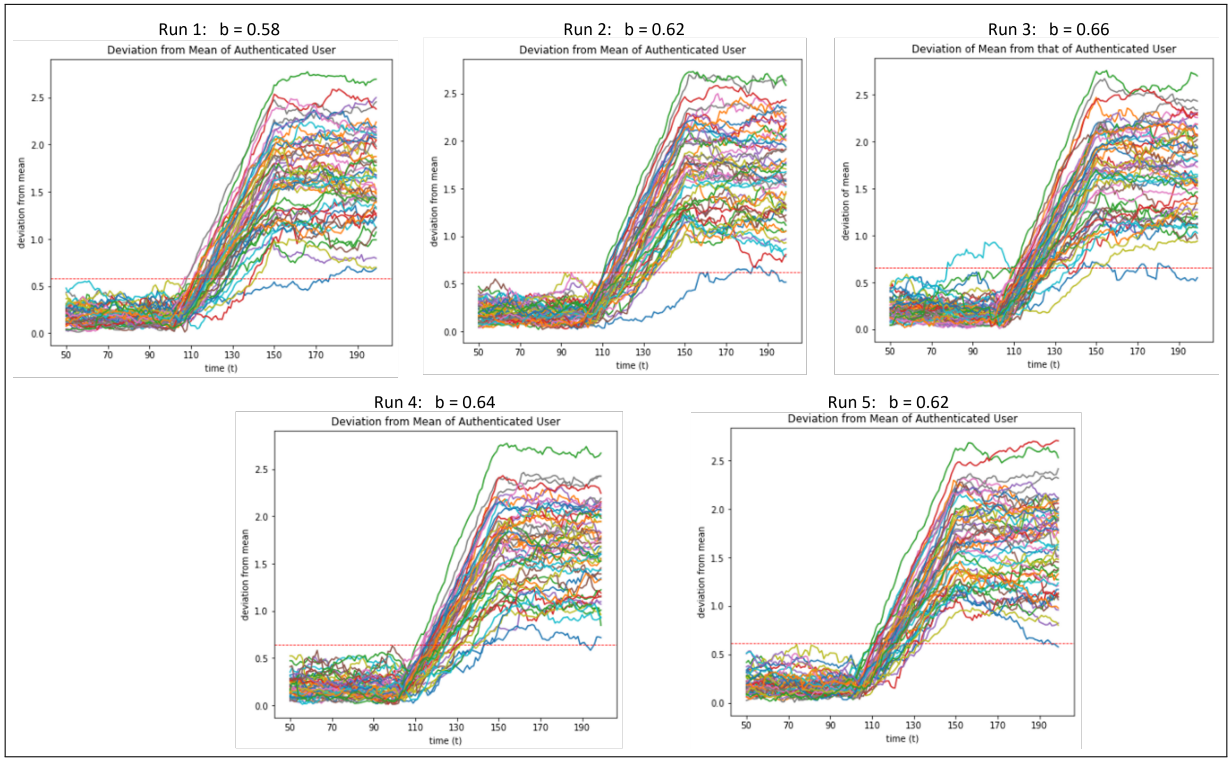


Notice that for this simulation, a threshold of $b = 0.58$ is ideal, in that it has no false positives and no false negatives.

Step 3: Perform cross-validation to tune b

We now perform cross-validation to tune b. We do this by running four (4) more simulations for a total of five (5). Figure 3.3.1-6 shows the results.

Figure 3.3.1-6. Cross-Validation (5 Runs)



Step 4: Take b to be the average b from all five (5) simulation runs

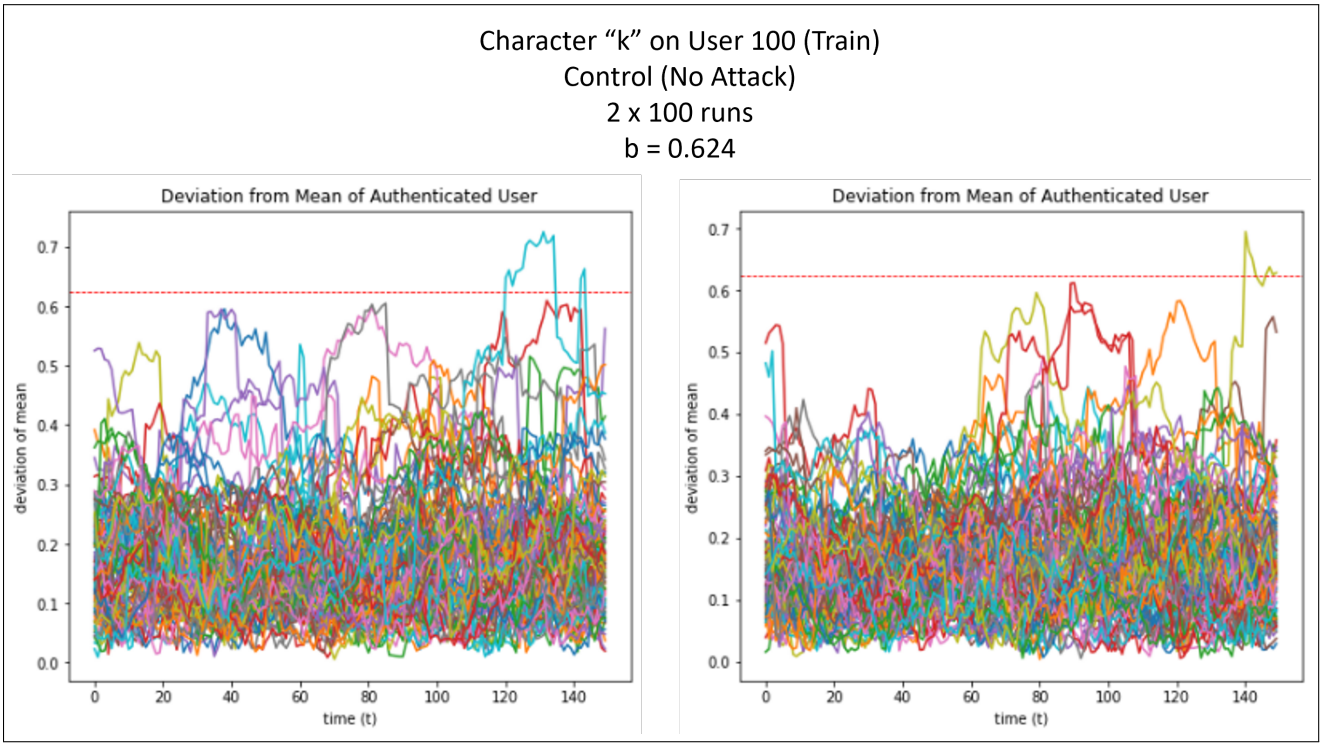
Each simulation run has a different optimal threshold b. We therefore take our optimal b to be the average of all 5 runs.

$$
\begin{aligned}
b &= \frac{1}{5}\sum_{i=1}^{5} b_i \\
&= \frac{1}{5}\left(0.58 + 0.62 + 0.66 + 0.64 + 0.62\right) \\
&= 0.624
\end{aligned}
$$

Step 5: Perform a control experiment where no attack occurs

For completeness, we perform 100 runs of a control experiment wherein no attack occurs. That is, we feed the model 200 data points all from the current authenticated user. The goal is to double-check for false positives. We do this on the training set. Figure 3.3.1-8 shows the results.

Figure 3.3.1-7. Control Experiment (100 runs)

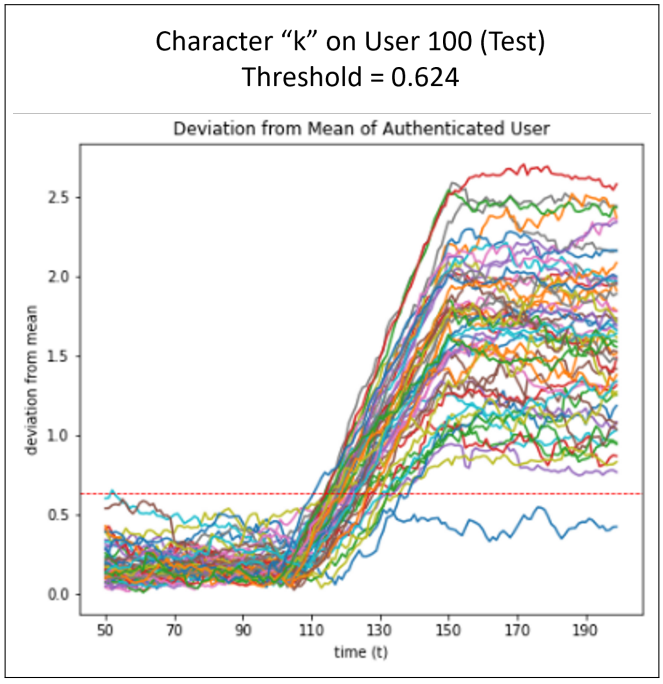


Only one run out of 100 resulted in a false positive. As a 1% false positive rate is an acceptable FRR, we do not update our threshold b.

Step 6: Use the learned threshold b and run the simulation using the test set

Now, using this threshold $b = 0.624$, we run a simulation on the Test set, to evaluate how our model performs. Figure 3.3.1-7 shows the results.

Figure 3.3.1-7. Test Set Simulation

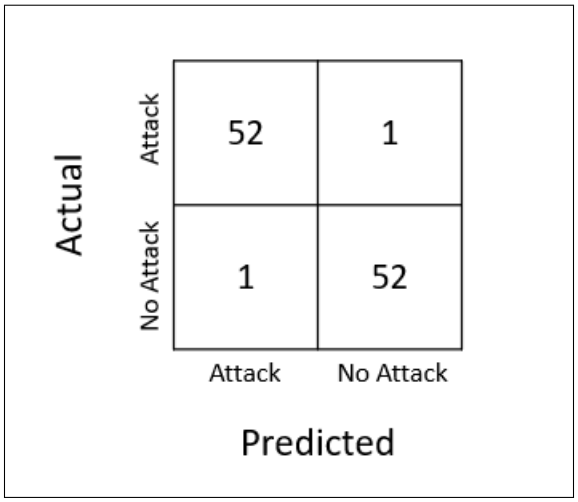


We see the model performs very well, with only one false rejection and only one false acceptance.

Step 7: Construct confusion matrix and evaluate performance of our model

Figure 3.3.1-8 shows the confusion matrix for User ID 100.

Figure 3.3.1-8. Confusion Matrix (User 100)



This gives a False Rejection Rate of $1/53 = 1.8\%$ and also a False Acceptance Rate of $1/53 = 1.8\%$.

Although we only showed results for one user for brevity, we did perform the test for all other users, with similar success. We tuned b for each user, such that their FAR and FRR were equal (1.8%). Thus, the average FAR and FRR for the model as a whole is 1.8%, and the **EER is also 1.8**%, which is likely acceptable for our use case. An adopting organization could tune the thresholds for each user to either have fewer false rejections or fewer false acceptances, depending on their particular security needs.

Therefore we conclude that the GLR approach when applied to post-authentication change detection works very well.

Discussion & Future work

The dataset that we had available was quite limited. In a real-world scenario, an organization would have the ability to have all its user go through a more thorough and rigorous "training exercise", wherein each user would type long paragraphs of various texts at different times, and their metadata recorded. This could be used to perform a much more robust model optimization activity, wherin the optimal set of characters could be determined. A trade study could be performed to evaluate model performance vs computational load. Perhaps models for only one or two letters or digrams (pairs of letters) would be needed to achieve good success, and attackers could be caught more quickly if those letters/digrams frequently occur in the target language. For example, in the English language, the two most common letters are 'E' and 'T', and the most common pairs of letters are 'TH', 'HE', and 'IN'. Training models on these characters would ensure relevance during actual use. (I.e., you would not want to train a model on letters or pairs of letters that are used so infrequently in practice that they would not be typed enough to be able to reasonably perform change detection.)

4 Evaluation and Final Results

In this section we compare the overall results of the False Rejection Rate (FRR) and the False Acceptance Rate (FAR) for all models, except the and the GLR Change Detection method as these were for completely different use cases and thus would not be relevant to directly compare to the other models. Also, Naive Bayes is not included in this comparison, since it was determined to be an inappropriate technique for this use case due to the significant dependence between many of the features. Furthermore, the Eigen-Password method was not explored past our preliminary findings dues to poorer performance, so it will be excluded from the comparison of the more promising candidates.

K-Means was found to not be effective at distinguishing between individual users. It did however seem to be able to identify similar groups of like users. Therefore it could still be useful in an organization where a supervised approach could not be undertaken. If the K-Means model determined that the individual entering the password came from a completely different group of users, access could still be denied, and benefit could still be realized. However, this model could do nothing if the attacker had similar typing behaviors to the current user, as it would group them in the same cluster.
The GLR method for change detection was shown to work well, with an FRR rate equal to its FAR rate at 1.8%.
Table 4-1 summarizes the results from all other models for the Initial Authentication use case.

Table 4-1. Summary of Model Results

Evaluation Metric	**Logistic Regression**	**Neural Network (Multi)**	**Neural Network (Binary)**	**Random Forest (Multi)**
FRR	0.00	0.127	0.029	0.061
FAR	0.008	0.002	0.019	0.001
Overall Accuracy	0.992	0.874	0.981	0.964

If one goes purely by accuracy, Logistic Regression performs the best and has the lowest FRR. However, it did not have the lowest FAR, which may be of concern in high-security settings where denying unauthorized access to sensitive information is of the utmost importance. As stated previously in the Neural Network section, a robust Defense-in-Depth security posture would likely implement an Intrusion Detection framework that would would likely involve multiple of these models as well as many others. For example, using a model with a low FRR to determine whether to grant access to a user, with any identified unauthorized login attempts transmitted back to the organization where it could then be fed into one or more multi-class and binary models to determine whether the user was likely to have come from within the organization (potential insider threat) or from outside.

Additionally, perhaps the GLR post-authentication change detection method could dynamically implement a more strict threshold if the initial authentication model was less confident about its decision of having granted access.

No computer algorithm alone is incriminating enough to warrant prosecution against an individual. Yet these algorithms clearly could provide value to an organizations security team, giving them tips and cues on which to potentially act to further investigate any users flagged as insiders. And the GLR post-authentication method is a promising method for preventing continued use of stolen or unattended unlocked devices.

Regarding feature selection, we see that the 2nd order features were indeed important after all, and the first order features that have more to do with a single character are more important than those that are for *pairs* of letters.

Overall, we feel many of the methods presented in this survey could prove effective additional layers of security to organizations, particularly those that have no *behavioral*-based multi-factor authentication (MFA) capability. ■

References

[1] The Mobikey dataset: http://www.ms.sapientia.ro/∼manyi/mobikey.html.

[2] Yao Xie, Meng Wang, and Andrew Thompson, "Sketching for sequential change-point detection", in Global Conference on Signal and Information Processing (GlobalSIP), 2015, pp. 78–82. https://arxiv.org/pdf/1505.06770.pdf

[3] https://www2.isye.gatech.edu/∼yxie77/change_point_simple.pdf

[4] https://www.proofpoint.com/us/resources/threat-reports/cost-of-insider-threats

[5] https://machinelearningmastery.com/feature-selection-with-numerical-input-data

[6] https://www.recogtech.com/en/knowledge-base/security-level-versus-user-convenience

[7] Wang, Fei & Franco-Penya, Hector-Hugo & Kelleher, John & Pugh, John & Ross, Robert. (2017). An Analysis of the Application of Simplified Silhouette to the Evaluation of k-means Clustering Validity. 10.1007/978-3-319-62416-7.