IMPROVING KEYSTROKE AUTHENTICATION THROUGH COMBINING MACHINE LEARNING ALGORITHMS

A THESIS

Presented to the University Honors Program

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the

University Honors Program Certificate

Joshua Kasanjian

Spring 2020

I, THE UNDERSIGNED MEMBER OF THE COMMITTEE,

HAVE APPROVED THIS THESIS

IMPROVING KEYSTROKE AUTHENTICATION THROUGH COMBINING MACHINE LEARNING ALGORITHMS

BY

Joshua Kasanjian

Mehrdad Aliasgari, Ph.D. (Thesis Advisor)

Computer Engineering and Computer Science

California State University, Long Beach
Spring 2020

ABSTRACT

IMPROVING KEYSTROKE AUTHENTICATION THROUGH

COMBINING MACHINE LEARNING ALGORITHMS

By

Joshua Kasanjian

May 2020

Our society's reliance on technology is higher than ever before, which is why it is concerning that passwords are the sole authentication method in most software systems. Hackers are frequently leaking or cracking passwords, which enables others to gain unauthorized access to a system. Keystroke dynamics classification is a biometric authentication method currently being researched as a second layer of security at login. This involves analyzing the timing of each keystroke in a password to distinguish a genuine user from an impostor.

Researchers have conducted numerous studies on the topic using various statistical and machine learning models to classify the keystroke dynamics of password login attempts. However, variations in data collection and performance evaluation techniques across studies make it difficult to compare model performance. Our study aims to compare the top performing algorithms of multiple studies, using a robust dataset. Moreover, we propose that using multiple algorithms to predict at once, and making a classification based on the resulting majority vote, could result in higher performance.

We found that the random forest classifier alone obtained the highest average precision of 99.8% and the lowest average recall of 90.4%. The second most accurate classifier, SVM, showed a precision and recall of 94.1% and 94.7%, respectively, with the others similarly showing more balanced results. The top performer when using combinations of classifiers resulted in a precision of 97.7% and recall of 94.5%. We can conclude that the top majority vote combination classifiers are more accurate overall with more potential applications.

ACKNOWLEDGMENTS

This thesis would not have been possible without help from Jonathan Bazan. We wrote the source code for this project together throughout the semester, and we plan to continue expanding on this research in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

As technology continues to become increasingly relevant in our everyday lives, security risks rise as well. Sensitive user data is stored on the myriad of computer systems in use today, and passwords are the main form of user authentication being used. However, if an adversary gets a hold of a user's password, they then gain unauthorized access to the user's data and can use it for malicious intent. Login credentials are frequently exposed and exploited by attackers, and users are often unaware that their accounts have been compromised. This is a result of two factors. Firstly, as computing continues to advance and processors become more powerful, the cost of brute-force and birthday attacks decreases; this makes it easier for attackers to guess user passwords (Chester, 2015). Secondly, a significant percentage of systems are equipped with outdated security protocols with known vulnerabilities, enabling other attacks that can lead to data breaches and leaked user passwords. The 2020 Open Source Security Risk and Analysis report by Synopsis determined that 70% code that was audited for security vulnerabilities as a result of no longer being maintained was open source, meaning that a large portion of software in use has potential vulnerabilities that can be exposed (Synopsis, n.d.). Furthermore, according to a survey conducted by Google, two out of three users recycle their password across the various services they use, so having their password exposed from any one of their accounts can have devastating consequences (Google, 2019).

The problem lies in distinguishing between a genuine user and an impostor with knowledge of the genuine user's password. One method that is growing in popularity is analyzing the keystroke dynamics, or the typing habits, of a user. Every individual types in a unique

way; for instance, the timing between each keystroke when typing the same word is usually consistent. As a user types their password repeatedly to login to a service, their keystroke dynamics become increasingly regular. If an adversary were to gain access to a user's password and attempt to log in, they would not type the password with the exact same keystroke timing as the genuine user. As a result, keystroke dynamics can be used as a biometric to identify a genuine user from an impostor, adding a second layer of security for user authentication.

Researchers have explored numerous anomaly detection and machine learning algorithms to analyze password keystroke dynamics and classify attempts as either the genuine user or an impostor. While many proposed algorithms yielded high accuracy, researchers used varying methods for data collection, feature representation, and performance evaluation. Consequently, it is difficult to compare each proposed algorithm, making it challenging for software developers to identify the best solution they can integrate into their login system as an extra layer of security. A solution must be proven to be reliable if it is to be used to protect sensitive user data. In order to meet the European standard for access-control systems (EN-5013301), a detector must perform with a false negative rate of less than 1% with a false positive rate of at most 0.001% (CENELEC, 2002). This level of accuracy has not yet been achieved by any algorithms currently researched.

This thesis aims to compare the top-performing algorithms proposed by other researchers with a robust dataset and consistent performance metrics. Moreover, we are proposing using multiple algorithms simultaneously to classify datapoints based on the majority vote from each algorithm's prediction. We will use three algorithms to predict if the keystroke dynamics from an attempt are from the genuine user or an impostor, and make the final prediction based on what the majority of algorithms predicted. Currently, no other works have explored using multiple classifiers at once. We expect the combinations to accommodate for the individual strengths and shortcomings of each algorithm to yield higher accuracy than each classifier alone, with the hope to meet the EN-5013301 standards.

CHAPTER 2

BACKGROUND AND RELATED WORKS

Machine Learning Overview

Machine learning is a subfield of artificial intelligence that involves learning from data to make predictions. Traditionally, solving a problem consists of writing explicit instructions on how to process input to produce output. Using machine learning enables programs to learn the pattern of data to predict the output of a function, which is extremely useful for complex problems where explicit instructions are difficult or impossible to construct. A machine learning model is the program that learns from data through training, and outputs predictions given input. Models are also known as classifiers, and the input values used to train and test models are called features. Figure 1 illustrates a high-level overview of this process.

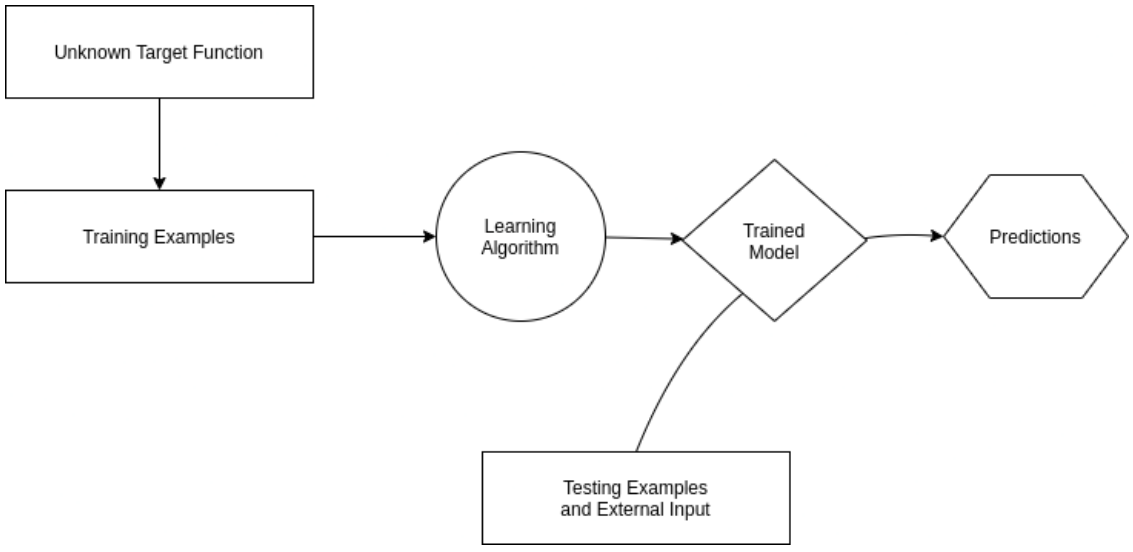

FIGURE 1. Basic overview of creating a machine learning model

Each component plays a role in constructing a machine learning model. This thesis aims estimate the unknown target function, "given the keystroke dynamics of a password attempt, was it typed by the genuine user?" The features of our keystroke dynamic dataset are made up of the time in-between each keystroke, which will be described in more detail in Chapter 3. Our training examples consist of the keystroke dynamics of the genuine user, as well as from impostors, which are used to teach the learning algorithm what the genuine user's keystroke dynamics look like. This data is passed in as (x, y), where x is the vector of training example features, and y is the vector of class labels for each example in numerical form. For instance, for datapoint n, $y_n = 1.0$ if it was from the genuine user and $y_n = -1.0$ if it was from an impostor. This is an example of supervised learning, where the class labels are known at training time. A model is considered trained once it optimises its error rate by going through all the training examples. At this stage it can make predictions when given test or external data. The next time someone tries to log into the user's account, the model should be able to distinguish if the attempt was made from the genuine user or an intruder.

There are a few other factors to consider when constructing a machine learning model that can affect its accuracy, such as the data source and the hyperparameters selected. Hyperparameters are properties of a learning algorithm that control aspects of the learning process. Different algorithms have different hyperparameters that need to be set. Moreover, the data source for training examples should be representative of the real-world target function being estimated. This is important to avoid bias and overfitting, both which can show high training accuracy but suffer when testing with external data.

Because machine learning models excel at learning patterns given sufficient data, they are widely used in computer security. Anomaly detection is an important part of cybersecurity, as adversaries attempting to break into a system exhibit behavior that differs from the behavior of normal users. Machine learning algorithms are effective at detecting such anomalies in system behavior, and a plethora of researchers are exploring various types of algorithms to test which ones are the highest performers for this application.

Previous Works on Keystroke Dynamics Classification

Keystroke dynamics classification is a growing area of research today, especially with the use of machine learning algorithms. This area was first explored in 1977 to investigate whether users could be distinguished based on the way they typed their name (George E Forsen, 1977). Since then, an extensive amount of research in the area has been carried out.

Kevin S. Killourhy (2009) compared 14 anomaly detectors for password intrusion detection using keystroke dynamics. They noted that at the time, it was nearly impossible to compare anomaly detectors because of the inconsistent set of evaluation conditions across studies. As a result, they collected their own robust dataset and made it available to the public after their paper was published. While they mostly used statistical anomaly detectors, they found that classifying using the Manhattan (Scaled), Nearest Neighbor (Mahalanobis), and Outlier Count (z-score) were the top performing methods, with equal-error rates of about 10%. A meta-analysis of all the existing keystroke biometric systems that have been tested was published in 2017 (Md Liakat Ali1, 2017). It compared many properties for each model in addition to the varying performance metrics different researchers had used.

Hani Jawed (2018) explored the use of biometric keystroke authentication in the domain of mobile devices. What makes this study unique is that they utilized the area on the screen that the user's fingers covered when typing there password as an additional feature along with the tap dynamics. They obtained an average accuracy, precision, and recall of 98.2% from random forest classification, and tested a series of other machine learning algorithms including k-nearest neighbor, support vector machine, logistic regression, and Naive Bayes classifier.

CHAPTER 3

METHODS

This chapter provides a high-level overview of the methods we used to approach this problem and process our data.

Data Source

We chose to use the dataset collected by Kevin S. Killourhy (2009) at Carnegie Mellon University, consisting of 51 subjects typing a password 400 times each. We selected this dataset because it provides an ample amount of datapoints and subjects, with each datapoint containing a set of representative features suited for keystroke dynamics analysis. Moreover, the data was made publicly available for other researchers to use.

All subjects typed this same password obtained from a public password generator: `.tie5Roanl`. This represents a typical strong password as it contains more than 7 characters, a capital letter, a number, and punctuation. The raw timing data (consisting of timestamps for each key event) of each password repetition was collected using a Windows program, which prompted subjects to retype the password if entered incorrectly. The same external keyboard was used for all subjects to ensure environment consistency and timestamp accuracy. The raw timing data was transformed into 31 features representing the keystroke dynamics of a repetition, which are described below.

- H.*key* : hold time for *key*
- DD.*key1*.*key2* : keydown-keydown time for the pair of keys (i.e., the time from when *key1* was pressed to when *key2* was pressed)

- UD.*key1.key2* : keyup-keydown time for the pair of keys (i.e., the time from when *key1* was released to when *key2* was pressed)

Each of the 400 repetitions consisted of a set of all 31 features, specifying the typing timing between every keystroke in the password `.tieRoanl`.

Algorithm Selection

In order to determine which machine learning algorithms to implement, we reviewed previous works and observed which algorithms performed the highest on keystroke dynamics classification. Many studies used different performance metrics for algorithm evaluation, which we acknowledged when comparing findings (Md Liakat Ali1, 2017). We also considered other factors that could affect the performance of an anomaly-detector algorithm. Variations in the dataset such as size, collection method, class balance, and method of partitioning all play a role in how accurate an algorithm classifies. Moreover, the accuracy of a model is affected by the hyperparameter values, which are assigned to try and yield the highest accuracy on the training dataset. We noted these values in studies that included them.

We required at least three machine learning algorithms in order to test a majority vote classifier, so we decided to implement the following algorithms: k-nearest neighbor (KNN), logistic regression, random forest, and support vector machine (SVM). Building four classifiers allowed us to test four distinct combinations to compare results, and after comparing the classifiers studied by others, this set of four appeared promising. For instance, KNN was the top-performing algorithm obtained by Kevin S. Killourhy (2009), with SVM obtaining the fourth highest accuracy out of 14 classifiers. Hani Jawed (2018) showed that SVM obtained the highest accuracy, followed by random forest and logistic regression. Therefore, we considered them all good candidates to evaluate and combine in a majority vote system.

Approach

Our research plan consisted of five steps:

1. Partition data into testing and training sets for each subject

2. Train and save models for every subject with hyperparameter tuning

3. Evaluate the performance of each model on the testing dataset when classifying alone

4. Evaluate the performance of every combination of three models when classifying the testing dataset based on the majority vote of predictions (i.e., select the most agreed-upon value as the classification)

5. Compare results and determine the top-performing classifiers

Hyperparameters

As aforementioned, hyperparameters have an impact on the performance of machine learning algorithms because they specify certain properties on how a model will learn. They must be set before training, but their optimal values depend on the problem. Hyperparameter values are either predetermined from human insight, or determined by a process called hyperparameter tuning, which involves testing different combinations of values in order to observe which combination yields the highest accuracy on the dataset.

We considered using the same hyperparameter values that other researchers used in their studies; however, because other datasets varied from ours, we decided that tuning our hyperparameters was the best approach when finding the optimal values for each model. Moreover, because every subject has their own set of training and testing data, we opted to tune the hyperparameters of a given model for each subject, as opposed to using the same values for the same model across all subjects. As a result, every model for a given subject had their own set of values to ensure optimal performance.

A widely used approach for finding the optimal hyperparameters is the use of a grid search. A grid search accepts a series of potential values for each parameter, and performs an exhaustive search (brute force) with all combinations of parameters. When tuning an SVM model, for example, if we wanted to test 3 different kernels such as $[linear, RBF, sigmoid]$ and all C values from the range of $[1, 1000]$, then a total of 3,000 different combinations need to be tested and evaluated. As a result, a grid search tends to be computationally ex-

pensive as the number of parameters being tested and values contained in these parameters increases. Altogether, performing an exhaustive search provides a means of obtaining an optimal solution with the best set of hyperparameters (Chih-Wei Hsu and Lin, 2016).

Logistic Regression

Logistic regression is used to predict binomial outcomes. A logistic regression classifier returns the probability that a given set of features x belongs to one of two classes. The logistic function θ, also known as sigmoid, is defined as follows:

$$\theta(s) = \frac{e^s}{1 + e^s}$$

The output of the sigmoid function is between 0 and 1, allowing it to be interpreted as a probability. Figure 2 shows the graph of the sigmoid function and the decision boundary of 0.5. In this instance, if the function returns a probability of 0.5 or greater given a set of keystroke features, that datapoint is classified as the genuine user; otherwise, it is classified as an impostor.

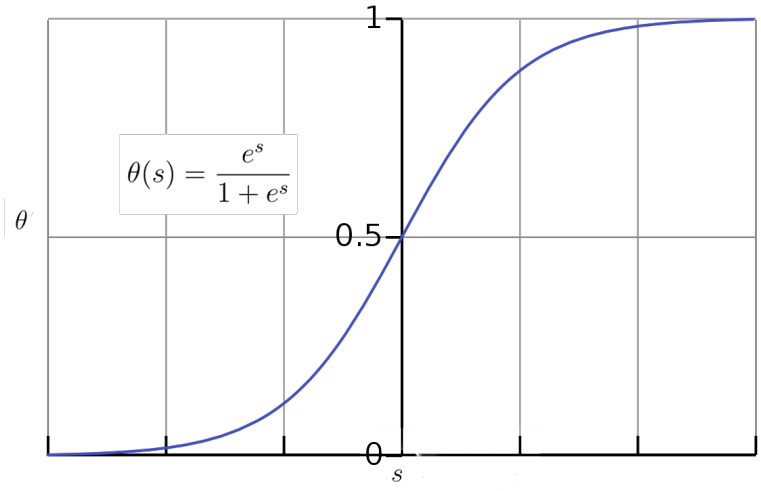

FIGURE 2. Graph of the logistic regression function

This equation is used to calculate the probability of a certain class given a feature vector:

$$P(y|x) = \theta(y * w^T x)$$

where x is the features vector, w is the weights vector, and y is the numeric class label you are testing for. Training a logistic regression model means finding the optimal weights w that minimize the error measure $E_{in}(w)$, defined as follows:

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} ln(1 + e^{-y_n w^T x_n})$$

Gradient descent is used during training to adjust w in order to minimize $E_{in}(w)$. Each iteration starting from $t = 0$, the gradient ΔE_{in} is calculated:

$$\Delta E_{in} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n x_n}{1 + e^{y_n w^T(t) x_n}}$$

The weights w are updated each iteration:

$$\Delta w = -\eta \Delta E_{in}(w)$$

The hyperparameter η is the learning rate, and max_iter specifies the maximum number of training iterations t that will occur during training. By the end of training, the optimal w that minimizes E_{in} is found. Tuning the hyperparameters will reduce the likelihood of overfitting, which can help improve testing accuracy (Stephan Dreiseitl, 2003).

K-Nearest Neighbor

The K-nearest neighbors algorithm classifies datapoints based on their similarity. During the training phase, the model memorizes all training examples and labels. When predicting the class of a new datapoint, the model finds the k closest neighboring datapoints. It then looks at the majority vote from the closest neighbors, and classifies the new datapoint accordingly. KNN makes the assumption that members of the same class share similar features and are in close proximity.

Similarity/proximity is measured using a distance metric. The KNN model that Kevin S. Killourhy (2009) tested as their highest performer used the Mahalanobis distance. However, we found that using the Minkowski distance metric yielded higher performance for our particular data partitions. The Minkowski distance between two points A and B is calculated as follows:

$$(\sum_{i=1}^{n} |A_i - B_i|^p)^{\frac{1}{p}}$$

The value of k is another hyperparameter that affects the accuracy. Figure 3 illustrates how a two-class KNN classifier would behave when attempting to classify a point when $k = 3$. The red and blue points were memorized by the model, and the circle encompasses the three nearest neighbors based on the distance metric. In this case, the point would be classified as blue, as two out of the three nearest neighbors have a blue label.

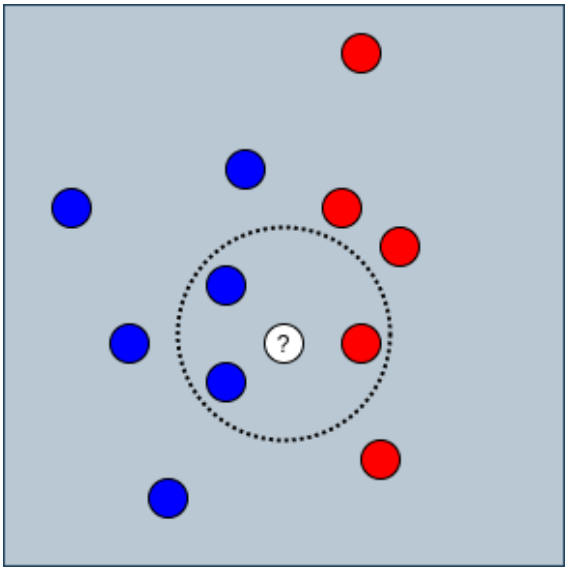

FIGURE 3. Example of a KNN classifier with $k = 3$

Random Forest

A powerful and widely used algorithm, random forest (RF) is a supervised learning algorithm that produces a series of decision trees, creating a forest that utilize ensemble

learning. In contrast to classification and regression trees, RF induces more randomness into the construction of the decision trees and follows a different approach for selecting which features to split a node with. Typically, the trees are trained independently and the predictions of the trees are combined through averaging to get the final prediction of the forest as described by Misha Denil (2013). A random forest initialzer accepts an input vector x_i and uses a random vector Θ_k for the k^{th} tree that does not have any correlation with previous vectors, to produce a set of classifiers $h(x, \Theta_k)$. A lot of the power in random forest classification lies in the margin function. Given $h(x_i)$, we sample a random amount of classifiers that ultimately yield a margin function show below:

$$mg(X, Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j)$$

in which the margin function is a metric for how the average number of votes cast compares with other available classes. The generalization error is defined by:

$$PE^* = P_{X,Y}(mg(X, Y)) < 0$$

(Misha Denil, 2013)

Breiman (1999) illustrates that as the number of tress goes to infinity the the generalization error tends to converge to an arbitrary value, meaning that even if we increase the number of trees exponentially, the model will not be overfitting after training.

$$PE^* \rightarrow P_{X,Y}(P_\Theta(h(X, \Theta) = Y) - max_{j \neq Y} P_\Theta(h(X, \Theta) = j) < 0)$$

Figure 4 depicts two trees of different depths sampled from a random forest that achieved high accuracy while not overfitting.

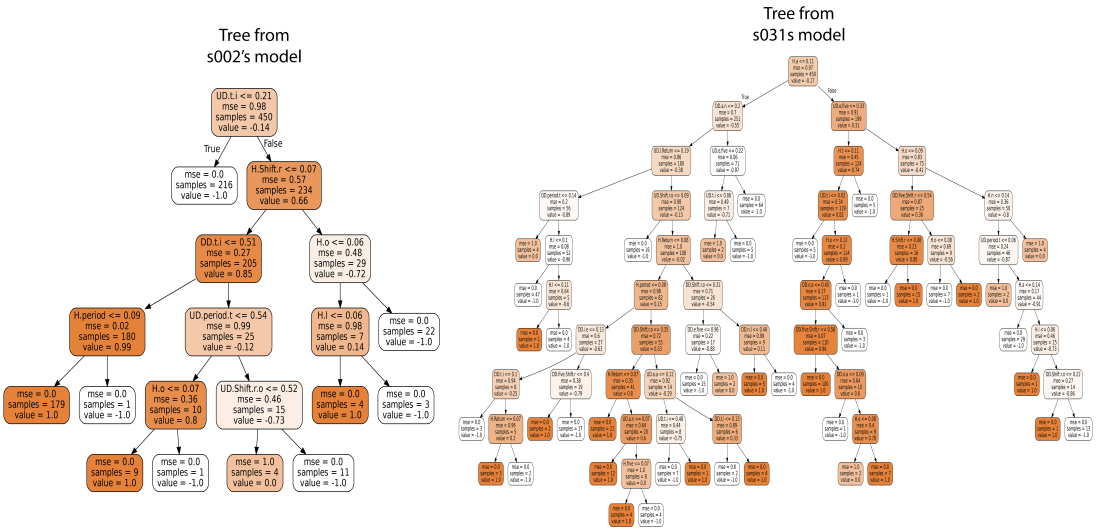

FIGURE 4. Example of trees produced by two random forest classifiers

Support Vector Machine

Another popular machine learning algorithm used for data classification is the support vector machine (SVM). At its core, a support vector machine attempts to optimize data-label pairs in the format (X_i, y_i) for $i = 1, .., n$, where $x_i \, \epsilon \, R^n$ and binary labels $y \, \epsilon \, (-1, 1)$. SVMs training optimize the following equation (Chih-Wei Hsu and Lin, 2016):

$$\begin{aligned} \underset{w,b,\xi}{min} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T \phi(x_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 1 \end{aligned}$$

Moreover, a common issue that arises when dealing with datasets both small and large, is that in the case of binary classification, datapoints may not be linearly separable. Fortunately, SVM inherently utilizes a kernel mapping trick which takes in vectors X_i and maps them to higher dimensions using the function $\phi(x_i)$. This produces a kernel function of the form

$$\text{Linear Kernel: } K(X_i, X_j) = \phi(x_i)^T \phi(x_j)$$

Similarly, other kernel functions that are widely popular such as sigmoid, polynomial, and radial basis function (RBF), provide a means of mapping to a higher dimension and producing a linearly separable hyperplane. These kernels may include additional parameters such as γ, d (dimension), and r (radius). However, the hyperparameter C requires a vast amount of consideration to fine-tune, as it is the penalty term when the function missclassifies a datapoint. Another hyperparameter used is γ which can be thought of as the spread of the decision boundary. Both of these parameters can be found in the widely used the RBF kernel, and Figure 5 illustrates the variations they cause. As we increase the γ in our function with C held constant, the function tends to undergo overfitting to the extent that some data-points may have there own islands (decision boundaries) as shown in Figure 5.4. On the other hand, if γ is held constant and the penalty variable is increased, we see the decision boundary try to minimize the amount of missclassified points while simultaneously experiencing low bias and high variance, which can ultimately lead to overfitting once again, as shown in Figure 5.2. These hyperparameters must be tuned in order to find the values that yield the highest performance given the subject's datasets.

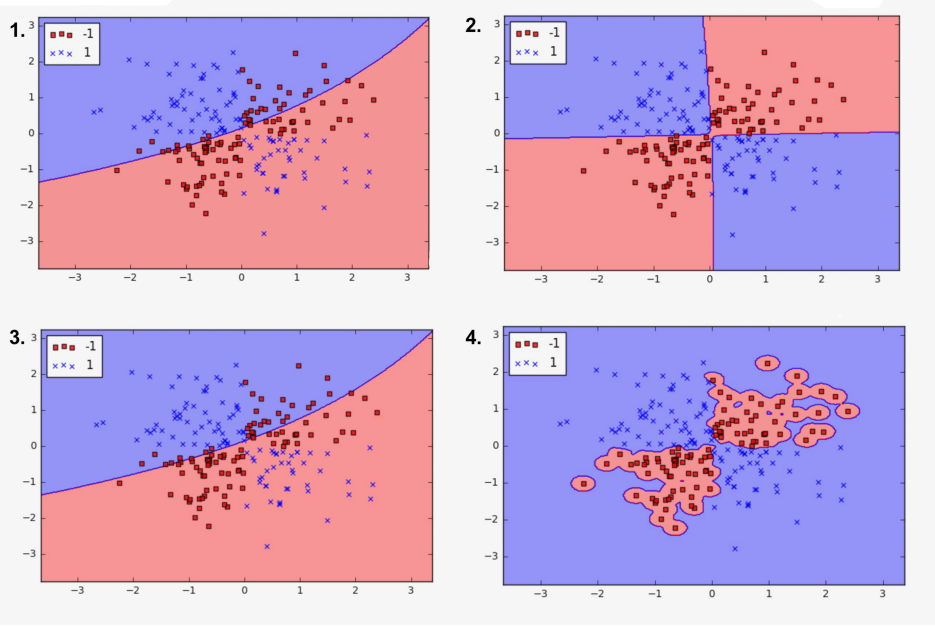

FIGURE 5. SVM with varying hyperparameters

Combining Models for Majority Vote System

Each combination of three models were evaluated with the same performance metrics as a single classifier. The prediction of each combination was the majority vote of each model's predictions. For instance, if at least two models predict a datapoint as the genuine user, then the combination classifies that datapoint as the genuine user. Every datapoint was tested using the majority vote system in order to calculate the specified performance metrics.

Evaluation

We used the following metrics to evaluate the performance of a solution: false acceptance rate, false rejection rate, precision, and recall. We used each subject's testing dataset to calculate these metrics for every model.

The false acceptance rate (FAR) is the rate at which an impostor was misclassified as the genuine user, and the false rejection rate (FRR) is the rate at which a genuine user was misclassified as an impostor. Precision and recall are accuracy metrics used for analyzing two aspects of a model's performance. Consider a positive classification as the genuine user, and a negative classification as an impostor. Precision shows the proportion of positive predictions that were actual positives. It is calculated by the following equation:

$$Precision = \frac{TP}{TP + FP}$$

where TP is the number of true positives in the data, and FP is the number of false positives (equivalent to false acceptances) from a given model. On the other hand, recall shows the proportion of actual positives that were predicted correctly. It is defined by the following equation:

$$Recall = \frac{TP}{TP + FN}$$

where FN is the number of false negatives (equivalent to false rejections) from a given model (Course, 2020). We calculated these metrics for each subject by comparing the predictions

of each classifier to the actual data labels, and calculated the mean and standard deviation (SD) of each classifier's metrics across all subjects to evaluate overall performance.

Because keystroke authentication for passwords has security-focused applications, precision and FAR were valued more than recall and FRR when comparing classifiers. Consider a system that uses password keystroke authentication as a biometric when logging in. A false acceptance would mean that an impostor with knowledge of a password was classified as the genuine user, giving them access to potentially sensitive data on the system that they could use for malicious intent. A false rejection would require the genuine user to re-type their password until accepted. A model with high precision has a low FAR, while a model with high recall has a low FRR. Recall and FRR are important to consider when focusing on providing a pleasant user experience, as having to re-type a password can be an annoyance. For highly sensitive applications, however, the increase in the FRR may be worth it order to keep the FAR low and reduce the probability of unauthorized access to the system.

CHAPTER 4

EXPERIMENTS AND RESULTS

This chapter goes over our experimental design and implementation, followed by the results and a discussion. All our experiments were written in Python (3.8.2). We used the machine learning library Scikit-learn (0.23.1) for machine learning implementation and evaluation. We also used the Python libraries numpy (1.18.4), matplotlib (3.2.1), joblib (0.15), and ipython (7.15.0) for additional data processing and visualization functionality. Our programs were executed on a system running Ubuntu 20.04 with a 16-core AMD Ryzen 9 3950X CPU and 32GB of DDR4 RAM.

Data Processing and Visualization

We downloaded the dataset from Kaggle as a CSV file. Using the built-in Python csv module, we read in the data file and converted it to json format using the built-in json module for easier access. Here we removed the subject ID from the feature set and set the ID as the key to each entry in the json file.

We used matplotlib to visualize the difference in the keystroke dynamics between subjects by plotting all 400 repetitions of each subject. Figure 6 shows the graphs of four subjects. We can observe the difference in their typing signature for the same password, illustrating the variances in each user's keystroke dynamics that we are aiming to detect more effectively.

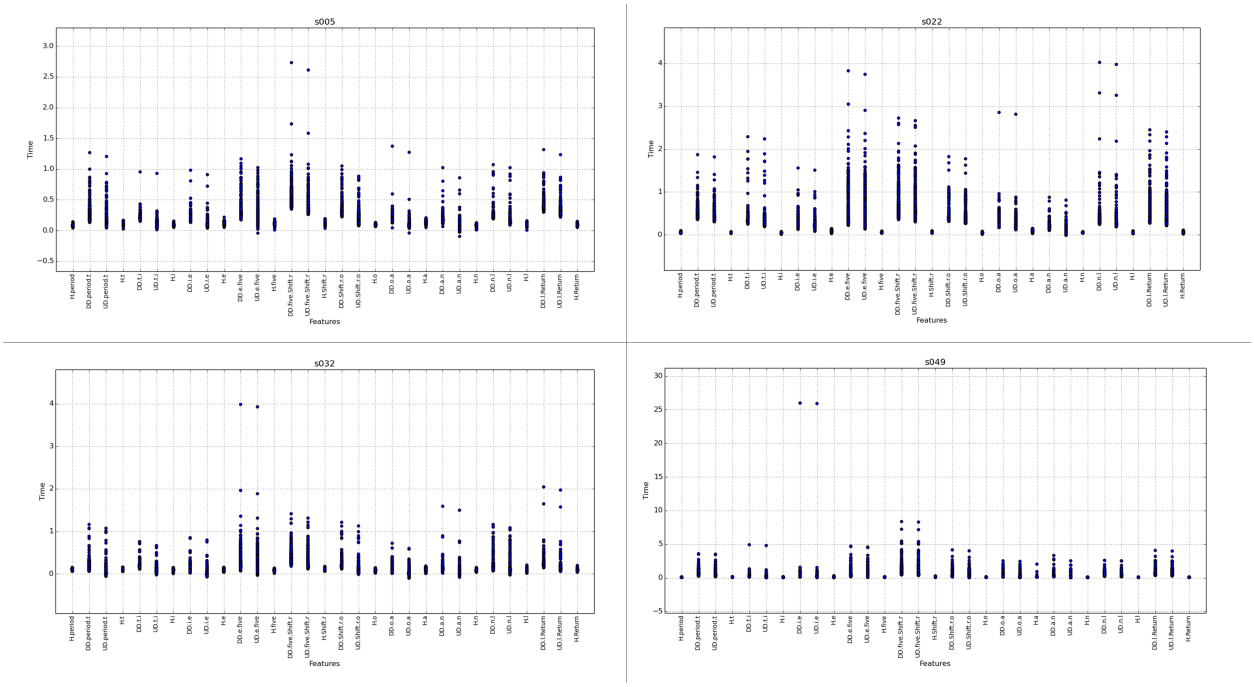

FIGURE 6. Visualization of features for subjects 05, 22, 32, 49

Algorithm Implementation

Hyperparameter Tuning

We used Scikit-learn's `GridSearchCV` module to tune hyperparameters by performing a grid search with all possible hyperparameters for each subject's models. `GridSearchCV` accepted a parameter distribution, training datapoints and labels, and the classifier, and returned the values of the parameter distribution that yielded the highest accuracy after trying every combination. We referenced at Scikit-learn's documentation of each model to construct the parameter distributions. The resulting set of best hyperparameter values were used to train each subject's models and saved to a .json file. Each model was saved to a file using the joblib library.

Logistic Regression

We used Scikit-learn's `LogisticRegression` module to implement our models and the following distribution for hyperparameter tuning: `{penalty:["l1", "l2", "elasticnet", "none"], C:numpy.logspace(-4, 4, 20), solver:["lgfgs", "newton-cg", "liblinear", "sag", "saga"], max_iter:[100, 1000, 2500, 5000]}`.

K-Nearest Neighbors

We used Scikit-learn's `KNeighborsClassifier` module to implement our models and the following distribution for hyperparameter tuning: `{leaf_size:list(range(1, 50)), n_neighbors:list(range(1, 30), p:[1, 2]}` .

Random Forest

We used Scikit-learn's `RandomForestRegressor` module to implement our models and the following distribution for hyperparameter tuning: `{n_estimators:[int(x) for x in numpy.linspace(200, 2000, 10)], max_features:["auto", "sqrt"], max_depth:[int(x) for x in numpy.linspace(10, 110, 11)], min_samples_split:[2, 5, 10], min_samples_leaf:[1 2, 4], bootstrap:[True, False]}` .

Support Vector Machine

We used Scikit-learn's `SVC` module to implement our models and the following distribution for hyperparameter tuning: `{kernel:["linear", "rbf", "poly"], C:[range(1, 1000, 10], degree:[1, 2]}`

Combinations for Majority Vote

1. Random forest + logistic regression + KNN
2. SVM + logistic regression + KNN

3. Random forest + SVM + KNN

4. Random forest + SVM + logistic regression

To test a combination on a designated genuine user, all three models for that subject were loaded. Each model made a prediction given a datapoint from the testing dataset. The datapoint was classified based of the majority vote of all model's predictions. This process was repeated for each subject's testing dataset in order to evaluate every combination of algorithms.

Evaluating Algorithms Alone

TABLE 1. Single Algorithm Performance

Algorithm	**Precision Mean**	Precision SD	**Recall Mean**	Recall SD	**FAR Mean**	FAR SD	**FRR Mean**	FRR SD
Random Forest	**99.675**	0.583	**90.336**	4.858	**0.144**	0.256	**4.937**	2.944
SVM	**94.118**	3.438	**94.705**	3.063	**2.619**	1.515	**2.375**	1.447
Logistic Regression	**93.428**	4.069	**94.264**	3.654	**2.937**	1.825	**2.575**	1.758
KNN	**92.702**	4.806	**94.047**	3.235	**3.259**	2.154	**2.641**	1.481

Table 1 shows the average precision, recall, FAR, and FRR (%) for each model across all subjects along with the standard deviations, in decreasing order of precision. The best average precision of 99.675% was obtained by the random forest classifier.

While it scored the highest on precision, it scored the lowest on recall. This means that has the lowest probability of accepting impostors into a system, but the highest probability of rejecting the genuine user from their own account. This illustrates the trade-off between precision and recall when evaluating model performance. If you increase the precision of a model by changing the decision boundary, the recall will decrease, and vice versa.

SVM obtained a more balanced set of performance metrics, with average precision and

recall rates of 94.118% and 94.705%, respectively. While this model has a higher FAR, the lower FRR would equate to a more pleasant user experience.

Evaluating Algorithm Combinations with Majority Vote Classification

TABLE 2. Combined Algorithm Performance

Combination	**Precision Mean**	Precision SD	**Recall Mean**	Recall SD	**FAR Mean**	FAR SD	**FRR Mean**	FRR SD
RF-LOG-KNN	**97.791**	2.365	**94.297**	3.484	**0.985**	1.043	**2.702**	1.812
RF-SVM-KNN	**97.691**	2.215	**94.432**	3.295	**1.028**	0.975	**2.623**	1.671
RF-SVM-LOG	**95.814**	2.852	**94.555**	3.542	**1.865**	1.266	**2.514**	1.772
SVM-LOG-KNN	**94.954**	3.289	**95.203**	3.070	**2.248**	1.458	**2.166**	1.465

Table 2 shows the average precision, recall, FAR, and FRR (%) for each combination of models across all subjects along with the standard deviations, in decreasing order of precision, where RF and LOG stand for random forest and logistic regression, respectively. While no combination reached a precision as high as that obtained by the random forest classifier alone, we did see an increase in precision without compromising on recall. The combination of random forest and KNN classifiers with logistic regression and SVM showed very similar performance. The RF-SVM-KNN combination in particular showed the most balanced high-performing results, with an average precision of 97.691% and an average recall of 94.432%. While the precision of this combination is 1.984% lower than that of the random forest classifier, the recall of the combination is 3.961% higher than that of the random forest classifier. When RF-SVM-KNN is compared to the most balanced single classifier, SVM, the combination increased the precision over SVM by 3.573%, while only showing a decrease of 0.273% in recall.

CHAPTER 5

CONCLUSION

Overview

All in all, using a combination of algorithms and taking the majority vote to classify yielded a better overall performance than using any single classifier alone. While we did not expect the random forest classifier to perform with such high precision, the majority vote combinations performed in-line with our hypothesis. Unfortunately, no classifiers scored high enough to qualify for the Eurpoean standard for access control systems (FRR under 1% and a FAR user 0.001%.). The random forest classifier with its extremely low FAR of 0.144% would be most effective if applied to systems with highly sensitive data. Systems like this must prioritize reducing the probability of unauthorized access to the system over user experience, as the low recall means that genuine users are rejected at a higher rate. On the other hand, the average user would likely opt-out of using a keystroke dynamic authentication with a FRR rate of 4.9037%, as having to re-type your password after many false rejections can be an annoyance. The combination of random forest, SVM, and KNN classifiers using the majority vote produced a high precision of 97.691%, while obtaining a good recall of 94.432%. This combination is the most practical choice we tested for potential integration into mainstream software today. It would be able to detect the majority of impostors while not inconveniencing the genuine user with frequent false rejections, which can add a much-needed extra layer of security to password authentication.

It is difficult to directly compare all our individual model results with those found in previous works due to the use of different performance metrics, datasets, and hyperparameters. However, we did observe that our random forest model performed significantly more

accurate than the random forest classifier tested by Hani Jawed (2018), which showed a precision of 90.57% and a recall of 89.20%. One possible explanation could be the extensive hyperparameter tuning we ran for each subject, whereas Hani Jawed (2018) only specified one hyperparameter, $n\_estimators = 70$, and used that value for all subjects. Notably for our KNN classifier, we used the Minkowski distance metric as opposed to the Mahalanobis metric used by the top-performing classifier obtained by Kevin S. Killourhy (2009). Even though we used the same base dataset, our KNN Mahalanobis tested at 20-30% lower accuracy. It may have something to do with the way in which Scikit-learn's `test_train_split` selects datapoints from each class. Furthermore, Kevin S. Killourhy (2009) used only positive data for training, whereas we used 200 positive samples and 250 negative samples, which could play a factor as to why we observed the Minkowski metric perform more accurately with data from both classes used in training.

Future Directions

In order to expand this research in the future, we plan on testing a wide array of different algorithms and combinations, such as neural networks, statistical classifiers, and more. This would leave us with a plethora of combinations to test and see how adding variety to the types of algorithms could affect accuracy. Additionally, having more algorithms to choose from would enable using more simultaneously to classify a datapoint. Taking the majority vote from the predicting of five or seven algorithms could have an affect on classifier performance. We could also take into account computational power required by each model. The classifiers would need to be efficient if they are implemented on software running on mobile phones to not hinder user experience with slow speeds.

We also plan to to look into using additional performance metrics during model evaluation such as F1 score and equal-error rate. We noticed many studies included these to further describe more aspects of a model's performance. We would also like to look into the frequency

that each algorithm votes on the correct side, and look into weighing votes differently based on the frequency. As right now all votes are weighted equally, but algorithms that predict correctly more often could be weighed more than others and potentially improve accuracy.

This work could have major implications for computer security, as many systems currently lack any additional layers of security beyond the user's password. With more of our lives going online, secure forms of authentication are essential to protect our data. After testing more combinations, we hope to find some that comply with the EN-5013301 standard that can be widely adopted in software systems across all disciplines.

REFERENCES

REFERENCES

Breiman, Leo (1999). *Random Forests - Random Features*. Technical Report.

CENELEC (2002).

Chester, John A. (2015). *Analysis of Password Cracking Methods and Applications*. URL: https://pdfs.semanticscholar.org/8f62/7279286100d10a45e4625f56b6ff43231513.pdf.

Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin (2016). "A Practical Guide to Support Vector Classification". In: *Department of Computer Science, National Taiwan University*.

Course, Google Machine Learning Crash (2020). *Classification: Precision and Recall*. URL: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

George E Forsen Mark R Nelson, Raymond J (1977). *Personal Attributes Authentication Techniques*.

Google (2019). *Online Security Survey*. URL: https://services.google.com/fh/files/blogs/google_security_infographic.pdf.

Hani Jawed Zara Zaid, Muhammaded M. Khan (2018). "Anomaly detection through keystroke and tap dynamics implemented via machine learning algorithms". In: *Turkish Journal of Electrical Engineering & Computer Sciences*, pp. 1698–1709.

Kevin S. Killourhy, Roy A. Maxion (2009). "Comparing Anomaly-Detection Algorithms for Keystroke Dynamics". In: *Dependable Systems Laboratory, Carnegie Melon University.*

Md Liakat Ali1 John V. Monaco1, Charles C. Tappert (2017). "Keystroke Biometric Systems for User Authentication". In: *Journal of Signal Processin Systems* 86.175-190.

Misha Denil David Matheson, Nando de Freitas (2013). *Narrowing the Gap: Random Forest In Theory and In Practice.* URL: http://proceedings.mlr.press/v32/denil14.pdf.

Stephan Dreiseitl, Lucila Ohno-Machadob (2003). "Logistic regression and artificial neural network classification models: a methodology review". In: *Journal of Bioinformatics* 35.6.

Synopsis (n.d.). *2020 Open Source Security and Risk Analysis Report.* URL: https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2020-ossra-report.pdf.