YOUR DEVICE MAY KNOW YOU BETTER THAN YOU KNOW YOURSELF-CONTINUOUS AUTHENTICATION ON NOVEL DATASET USING MACHINE LEARNING

Pedro Gomes do Nascimento, Pidge Witiak, Tucker MacCallum, Zachary Winterfeldt, Rushit Dave

Department of Computer Information Science, Minnesota State University, Mankato, Mankato, USA

ABSTRACT

This research aims to further understanding in the field of continuous authentication using behavioural biometrics. We are contributing a novel dataset that encompasses the gesture data of 15 users playing Minecraft with a Samsung Tablet, each for a duration of 15 minutes. Utilizing this dataset, we employed machine learning (ML) binary classifiers, being Random Forest (RF), K-Nearest Neighbors (KNN), and Support Vector Classifier (SVC), to determine the authenticity of specific user actions. Our most robust model was SVC, which achieved an average accuracy of approximately 90%, demonstrating that touch dynamics can effectively distinguish users. However, further studies are needed to make it viable option for authentication systems. You can access our dataset at the following link:https://github.com/AuthenTech2023/authentech-repo

KEYWORDS

Continuous Authentication, Machine Learning, Minecraft, Novel Dataset, Touch Gestures

1. INTRODUCTION

The current authentication methods, which are primarily implemented at entry points, can be problematic in numerous scenarios. Once a device is unlocked, it remains unlocked. The device can also be vulnerable to smudge attacks and keyloggers. In a world of ever-advancing technology, security of our devices is vital, and a proposed solution for authentication is the implementation of continuous authentication methods. Continuous authentication relies on the idea that a device could learn the user's behavioural biometrics patterns and identify whether an impostor or the actual user is using the device. Within this paper, we investigate continuous authentication methods rooted in machine learning. Our research is grounded in touch screen data acquired from 15 volunteers playing the popular game Minecraft, from which we extracted essential features. These features served as the foundation for training and assessing our machine learning models. The three types of machine learning models we used were: RF, KNN and SVC. In this paper, we make the following contributions:

1. We created a public touch dynamics dataset that records the actions of 15 users who played Minecraft on an Android device. The dataset is available at: https://github.com/AuthenTech2023/authentech-repo.

2. We trained and tested three models: KNN, SVC and RF. We then compared our outcome to results from previous works.

This paper is organized as follows: Section 2 reviews the related literature on this field and highlights the research gap. Section 3 describes the methodology, including the data collection and the models employed. Section 4 presents the results. Section 5 discusses the findings and compares them with existing studies. Section 6 acknowledges the limitations of the research and suggests directions for future work. Section 7 concludes the paper and summarizes the main contributions.

2. LITERATURE REVIEW

User authentication through touchscreen gestures has gained significant attention in recent years, fueled by the proliferation of smartphones and the need for robust security measures. The existing body of literature offers valuable insights into various models and methodologies employed to authenticate users based on their touch behavior. However, a notable gap in the literature lies in the limited exploration of individual user-centric authentication as opposed to the prevalent focus on userversus-imposter scenarios.

Several studies have addressed the broader context of touchscreen authentication, providing foundational knowledge of sensors commonly found in smartphones [1]. Achieving an 87% accuracy using a Siamese Neural Network architecture, this work sets the stage for more nuanced investigations [1]. Others have leveraged Random Forest to extract features and MultiLayer Perceptron for user authentication, achieving an impressive accuracy of 95.96% [2]. The exploration of different machine learning models has been a recurrent theme, with varied approaches such as K-means clusters and SVM [3], one-class SVM [4], and fusion models like SVM and GMM adapted from UBM [5]. However, the literature primarily revolves around generic touch tasks or predefined gestures without considering individual user nuances during free tasks. In contrast, this paper chooses an approach focusing on authenticating individual users within a constrained yet task-free setting. The choice of Minecraft as the application for data collection proves strategic, as it demands users to perform a diverse range of touchscreen gestures, which can garner better results [6][7][8]. The popularity and intuitiveness of Minecraft further enhance the applicability of the proposed solution.

Our selection of Support Vector Classification (SVC), k-Nearest Neighbors (KNN), and Random Forest as the machine learning models is informed by the success observed in previous studies [9][10][11]. Notably, KNN and Random Forest have demonstrated promising results, with average AUC values of 0.9091 and 0.9698, respectively [12]. The rationale behind this choice lies in the pursuit of models that showcase efficacy in touchscreen-based user authentication.

To contribute to the existing knowledge, we introduce a novel testing methodology by individually evaluating each user's touchscreen data rather than adopting the conventional user-versus-imposter paradigm. The 30-70 test split allows for comprehensive evaluation, ensuring a robust understanding of each model's performance for each user. Moreover, our decision to make the touchscreen data publicly available aims to foster collaboration and further advancements in the field.

Drawing inspiration from prior research [10][11], we carefully extracted features for model training, considering their efficacy in producing accurate and reliable results. Emphasizing adaptability to realworld applications, our work strives to bridge the gap between task-specific accuracy, which can lead to better results [13], and practical usability.

[14] presents a practical scheme for continuous user authentication on smartphones, incorporating a risk score mechanism. [15] proposed a global model eliminating the need for retraining on new users, addressing the challenge of computing resource constraints on mobile devices. These paper provide insights into the scalability and applicability of touchscreen authentication solutions in practical settings.

In summary, this literature review positions our study within the broader landscape of touchscreen based user authentication, highlighting the unique contributions of our approach. By focusing on individual user-centric authentication within the context of a popular application like Minecraft, and employing machine learning models with proven success, we aim to advance the understanding of touchscreen authentication and pave the way for practical, user-centric solutions.

3. METHODS

3.1. Data Collection

This research involved human subjects and required ethical training and approval. We completed the Collaborative Institutional Training Initiative (CITI) program, which covers various topics such as history and ethical principles, informed consent, privacy and confidentiality, students in research, and defining research with human subjects. We also obtained approval from the Institutional Review Board (IRB) to conduct the data collection on the Mankato State University campus with student participants.

3.2. Dataset

Table 1 – Metrics collected and their description.

Metrics	**Description**
Timestamp	Time of recorded touch event
X	X coordinate
Y	Y coordinate
BTN_TOUCH	Down - screen was just touched. Held - finger is on screen. Up - touch was just released.
TOUCH_MAJOR	Length of the major axis of the contact.
TOUCH_MINOR	Length of the minor axis of the contact.
TRACKING_ID	Identifies an initiated contact throughout its life cycle.
PRESSURE	The pressure on the contact area.
FINGER	Whenever there is a second finger on the screen it will have the value of “1”. Default value is “0”.

Data collection was performed using a python script that utilized the Android Debug Bridge (ADB) tool to access the touch screen metrics of the device. Our dataset consists of raw touch dynamics data from fifteen volunteers that played Minecraft for fifteen minutes on a Samsung A8 tablet. The metrics extracted from the device, presented in Table 1, include x and y coordinates, length and width of fingertip, touch pressure, when the screen is touched and released, and finger ids. The python script used for the data collection can be found at: https://github.com/AuthenTech2023/authentech-repo.

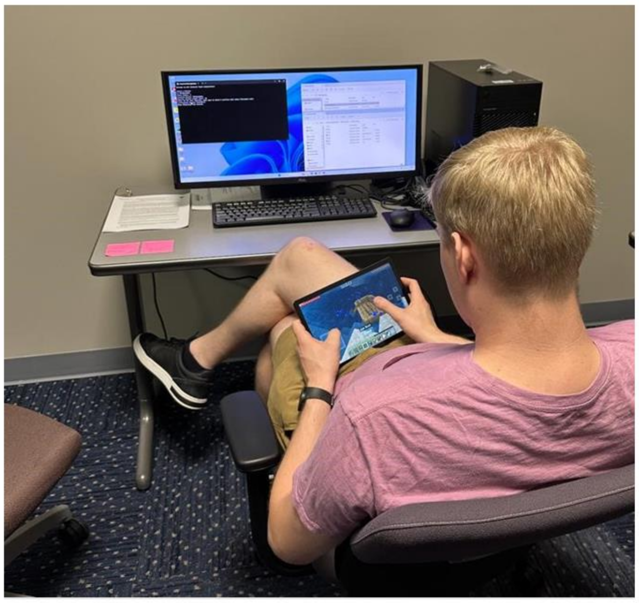


Figure 1 - Volunteer in data collection session

3.3. Data Cleaning and Processing

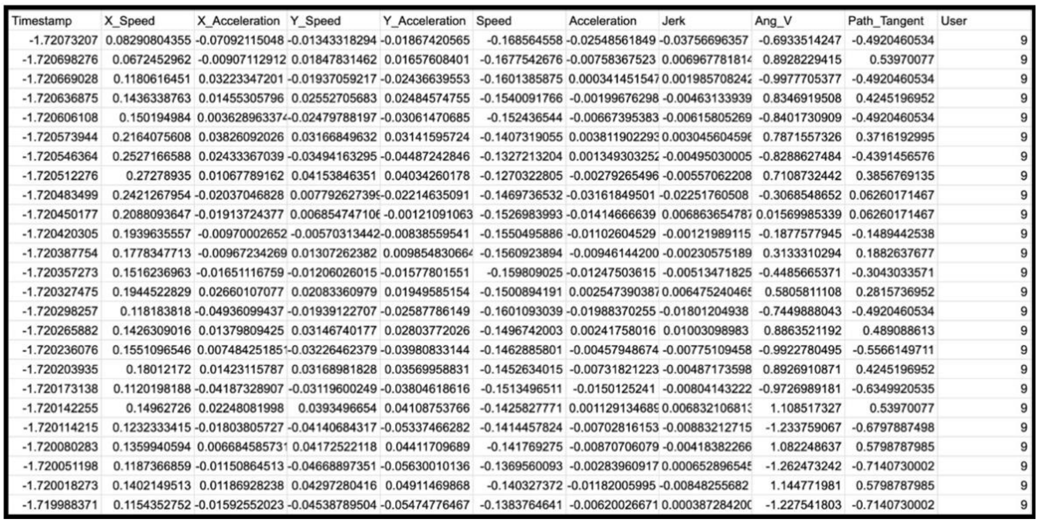
Data filtering techniques were applied to improve the quality and reliability of the dataset. Specifically, rows with default values of "-420" in either the X or Y coordinates were excluded to enhance the integrity of the data. The dataset was sorted by finger ID and timestamp to facilitate meaningful analysis. We converted all string values into numerical representations to ensure we could perform proper calculations with the data. Any rows containing missing (NaN) values were removed to ensure data completeness.

In addition to these filtering procedures, the numerical columns underwent a standardization process. This involved making the numbers more balanced and consistent. We used a tool from Sklearn called StandardScaler() that changes the numbers so that their average is 0 and their variation is 1. This makes it easier to compare them and see patterns. This standardization not only aids in achieving uniformity across features but also contributes to the interpretability of subsequent analyses.

These sequential steps, including data exclusion, sorting, conversion, and standardization, collectively contribute to the cleansing and enhancement of the dataset.

3.4. Feature Extraction

In the process of feature extraction from our cleaned and pre-processed dataset, we specifically selected the following key features: X_Speed and Y_Speed, representing the touch's instantaneous velocities along the X and Y axes; Speed, an aggregate speed measure derived from the X and Y individual components; X_Acceleration and Y_Acceleration, denoting the instantaneous accelerations along the horizontal and vertical directions; Acceleration, a comprehensive measure derived from the X and Y individual accelerations; Jerk, representing the rate of change of acceleration; Path_Tangent, capturing the touch's path angle; and Ang_V (Angular Velocity), providing insights into the touch's rotational dynamics.



Timestamp	X_Speed	X_Acceleration	Y_Speed	Y_Acceleration	Speed	Acceleration	Jerk	Ang_V	Path_Tangent	User
-1.72073207	0.08290804355	-0.07092115048	-0.01343318294	-0.01867420565	-0.168564558	-0.02548561849	-0.03756696357	-0.6933514247	-0.4920460534	9
-1.720698276	0.0672452962	-0.00907112912	0.01847831462	0.01657608401	-0.1677542676	-0.00758367523	0.006967781814	0.8928229415	0.53970077	9
-1.720669028	0.1180616451	0.03223347201	-0.01937059217	-0.02436639553	-0.1601385875	0.000341451547	0.001985708242	-0.9977705377	-0.4920460534	9
-1.720636875	0.1436338763	0.01455305796	0.02552705683	0.02484574755	-0.1540091766	-0.00199676298	-0.00463133939	0.8346919508	0.4245196952	9
-1.720606108	0.150194984	0.003628963374	-0.02479788197	-0.03061470685	-0.152436544	-0.00667395383	-0.00615805269	-0.8401730909	-0.4920460534	9
-1.720573944	0.2164075608	0.03826092026	0.03166849632	0.03141595724	-0.1407319055	0.003811902293	0.003045604596	0.7871557326	0.3716192995	9
-1.720546364	0.2527166588	0.02433367039	-0.03494163295	-0.04487242846	-0.1327213204	0.001349303252	-0.00495030005	-0.8288627484	-0.4391456576	9
-1.720512276	0.27278935	0.01067789162	0.04153846351	0.04034260178	-0.1270322805	-0.00279265496	-0.00557062208	0.7108732442	0.3856769135	9
-1.720483499	0.2421267954	-0.02037046828	0.007792627399	-0.02214635091	-0.1469736532	-0.03161849501	-0.02251760508	-0.3068548652	0.06260171467	9
-1.720450177	0.2088093647	-0.01913724377	0.006854747106	-0.00121091063	-0.1526983993	-0.01414666639	0.006863654787	0.01569985339	0.06260171467	9
-1.720420305	0.1939635557	-0.00970002652	-0.00570313442	-0.00838559541	-0.1550495886	-0.01102604529	-0.00121989115	-0.1877577945	-0.1489442538	9
-1.720387754	0.1778347713	-0.00967234269	0.01307262382	0.009854830664	-0.1560923894	-0.00946144200	-0.00230575189	0.3133310294	0.1882637677	9
-1.720357273	0.1516236963	-0.01651116759	-0.01206026015	-0.01577801551	-0.159809025	-0.01247503615	-0.00513471825	-0.4485665371	-0.3043033571	9
-1.720327475	0.1944522829	0.02660107077	0.02083360979	0.01949585154	-0.1500894191	0.002547390387	0.006475240465	0.5805811108	0.2815736952	9
-1.720298257	0.118183818	-0.04936099437	-0.01939122707	-0.02587786149	-0.1601093039	-0.01988370255	-0.01801204938	-0.7449888043	-0.4920460534	9
-1.720265882	0.1426309016	0.01379809425	0.03146740177	0.02803772026	-0.1496742003	0.00241758016	0.01003098983	0.8863521192	0.489088613	9
-1.720236076	0.1551096546	0.007484251851	-0.03226462379	-0.03980833144	-0.1462885801	-0.00457948674	-0.00775109458	-0.9922780495	-0.5566149711	9
-1.720203935	0.18012172	0.01423115787	0.03168981828	0.03569958831	-0.1452634015	-0.00731821223	-0.00487173598	0.8926910871	0.4245196952	9
-1.720173138	0.1120198188	-0.04187328907	-0.03119600249	-0.03804618616	-0.1513496511	-0.0150125241	-0.00804143222	-0.9726989181	-0.6349920535	9
-1.720142255	0.14962726	0.02248081998	0.0393496654	0.04108753766	-0.1425827771	0.001129134689	0.006832106813	1.108517327	0.53970077	9
-1.720114215	0.1232333415	-0.01803805727	-0.04140684317	-0.05337466282	-0.1414457824	-0.00702816153	-0.00883212715	-1.233759067	-0.6797887498	9
-1.720080283	0.1359940594	0.006684585731	0.04172522118	0.04411709689	-0.141769275	-0.00870706079	-0.00418382266	1.082248637	0.5798787985	9
-1.720051198	0.1187366859	-0.01150864513	-0.04668897351	-0.05630010136	-0.1369560093	-0.00283960917	0.000652896545	-1.262473242	-0.7140730002	9
-1.720018273	0.1402149513	0.01186928238	0.04297280416	0.04911469868	-0.140327372	-0.01182005995	-0.00848255682	1.144771981	0.5798787985	9
-1.719988371	0.1154352752	-0.01592552023	-0.04538789504	-0.05474776467	-0.1383764641	-0.00620026671	0.000387284200	-1.227541803	-0.7140730002	9

Figure 2 - User 9's tocuh features

4. METHODOLOGIES

In implementing our chosen ML models, we elected to train each model on all 15 users, and then tested the models on each user's data individually. This served as a challenge with accuracy, because it increased variability between users' results when the same model was deployed. 70% of each user's data was used to train the model, then the remaining 30% of each user's data was used for testing. The end-to-end process is shown below in Figure 3. To sum it up: We started by collecting raw data from 15 subjects, then process the data into the features we extracted. Then, we performed our traintest split, trained a model, tested it, and obtained our results. This process was followed for all three models.

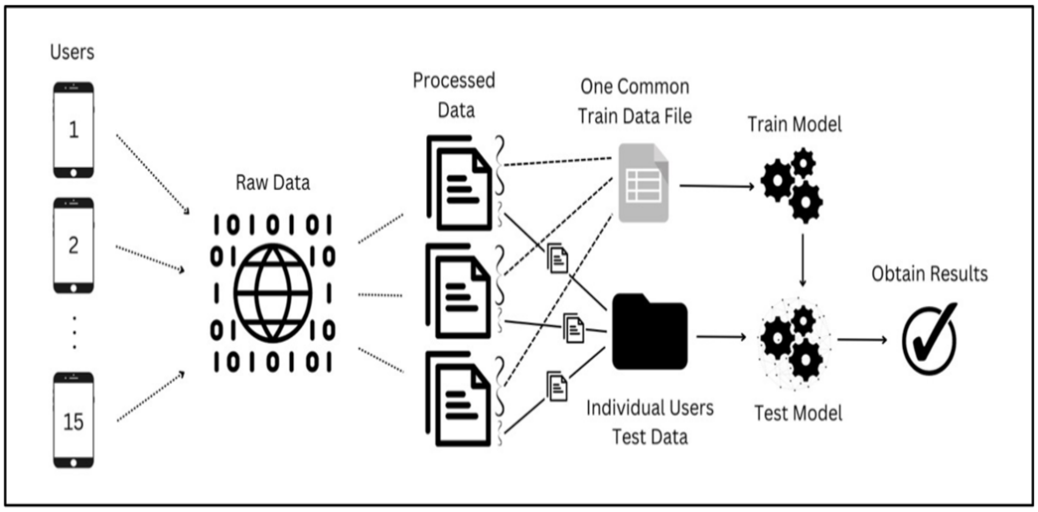
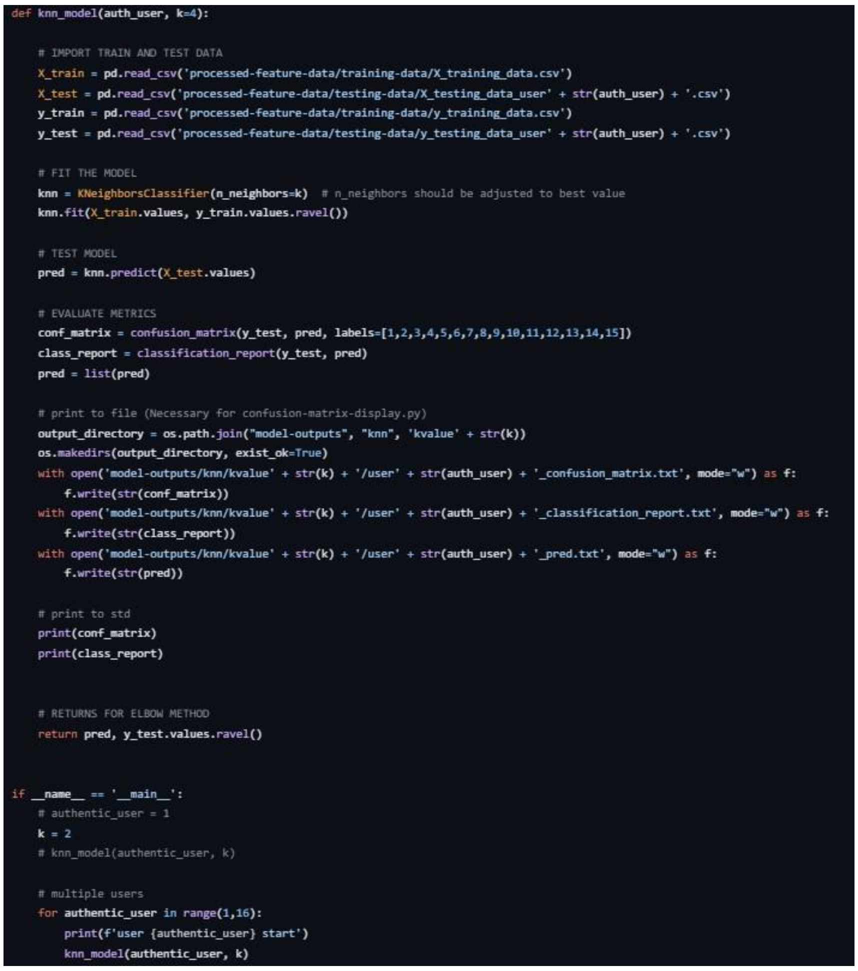


Figure 3 - End-to-End process of the experiment

4.1. The Models

4.1.1. K-Nearest-Neighbors

The KNN model classifies users based on the proximity of a data point to other classified data points. When training the model, labelled data points are plotted on a n-dimensional graph where n is the number of features in the data (one axis for each feature). Users are then classified according to the k closest data points, where k is an integer given to the model along with the testing data. If the k value is 3 and the 3 closest data points are of class 1, 1, and 2, the data point will be classified as class 1. When determining the best k value for our data set, we employed the use of the “Elbow Method”. The elbow method is a means of determining appropriate k values by running the model with a range of k values and plotting the error rate against the k values. From this graph the k value that looks to be at the moment of flattening error rates is deemed to be the best fit. To determine our k value, we used this method for each user’s dataset and chose the k value that was best fit for the most users (k=2).

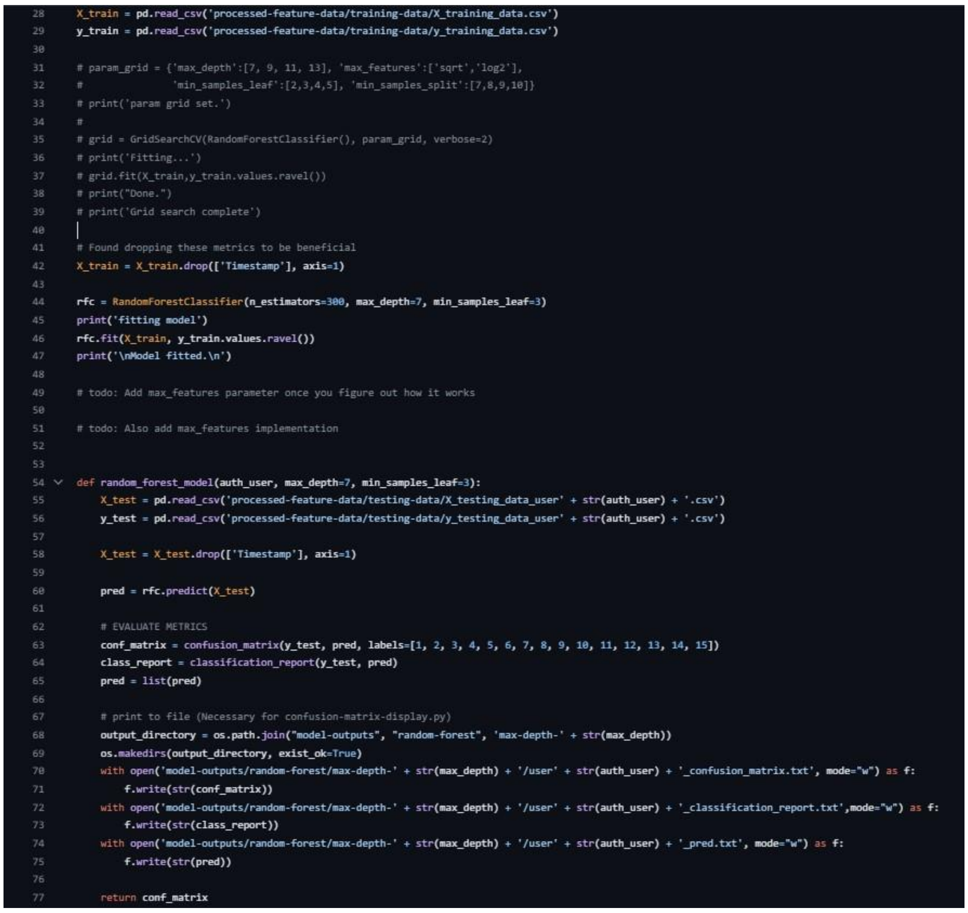


```python
def knn_model(auth_user, k=4):

    # IMPORT TRAIN AND TEST DATA
    X_train = pd.read_csv('processed-feature-data/training-data/X_training_data.csv')
    X_test = pd.read_csv('processed-feature-data/testing-data/X_testing_data_user' + str(auth_user) + '.csv')
    y_train = pd.read_csv('processed-feature-data/training-data/y_training_data.csv')
    y_test = pd.read_csv('processed-feature-data/testing-data/y_testing_data_user' + str(auth_user) + '.csv')

    # FIT THE MODEL
    knn = KNeighborsClassifier(n_neighbors=k)  # n_neighbors should be adjusted to best value
    knn.fit(X_train.values, y_train.values.ravel())

    # TEST MODEL
    pred = knn.predict(X_test.values)

    # EVALUATE METRICS
    conf_matrix = confusion_matrix(y_test, pred, labels=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
    class_report = classification_report(y_test, pred)
    pred = list(pred)

    # print to file (Necessary for confusion-matrix-display.py)
    output_directory = os.path.join("model-outputs", "knn", 'kvalue' + str(k))
    os.makedirs(output_directory, exist_ok=True)
    with open('model-outputs/knn/kvalue' + str(k) + '/user' + str(auth_user) + '_confusion_matrix.txt', mode="w") as f:
        f.write(str(conf_matrix))
    with open('model-outputs/knn/kvalue' + str(k) + '/user' + str(auth_user) + '_classification_report.txt', mode="w") as f:
        f.write(str(class_report))
    with open('model-outputs/knn/kvalue' + str(k) + '/user' + str(auth_user) + '_pred.txt', mode="w") as f:
        f.write(str(pred))

    # print to std
    print(conf_matrix)
    print(class_report)


    # RETURNS FOR ELBOW METHOD
    return pred, y_test.values.ravel()


if __name__ == '__main__':
    # authentic_user = 1
    k = 2
    # knn_model(authentic_user, k)

    # multiple users
    for authentic_user in range(1,16):
        print(f'user {authentic_user} start')
        knn_model(authentic_user, k)
```

Figure 4 – kNN Python code

4.1.2. Random Forest

The RF algorithm is based on the decision trees algorithm, which looks at all of the features, and sees when certain classifications are made based off of certain features paired together. In respect to this, a random forest uses multiple decision trees with random feature selection in attempts to reduce overfitting. We implemented the random forest algorithm using scikit-learn's RandomForestClassifier class in Python. Normalization, scaling, testing different features, and running a grid search were all undergone in efforts to attain the best results. The grid search revealed to us the optimal hyperparameters to use for each user. With this information, we recorded the mean and mode of these hyperparameters for all users. We even experimented with choosing different hyperparameters including max_depth, max_features, min_samples_leaf, and min_samples_split. The final hyperparameters we ended up utilizing were min_samples_leaf=3 and max_depth=6.

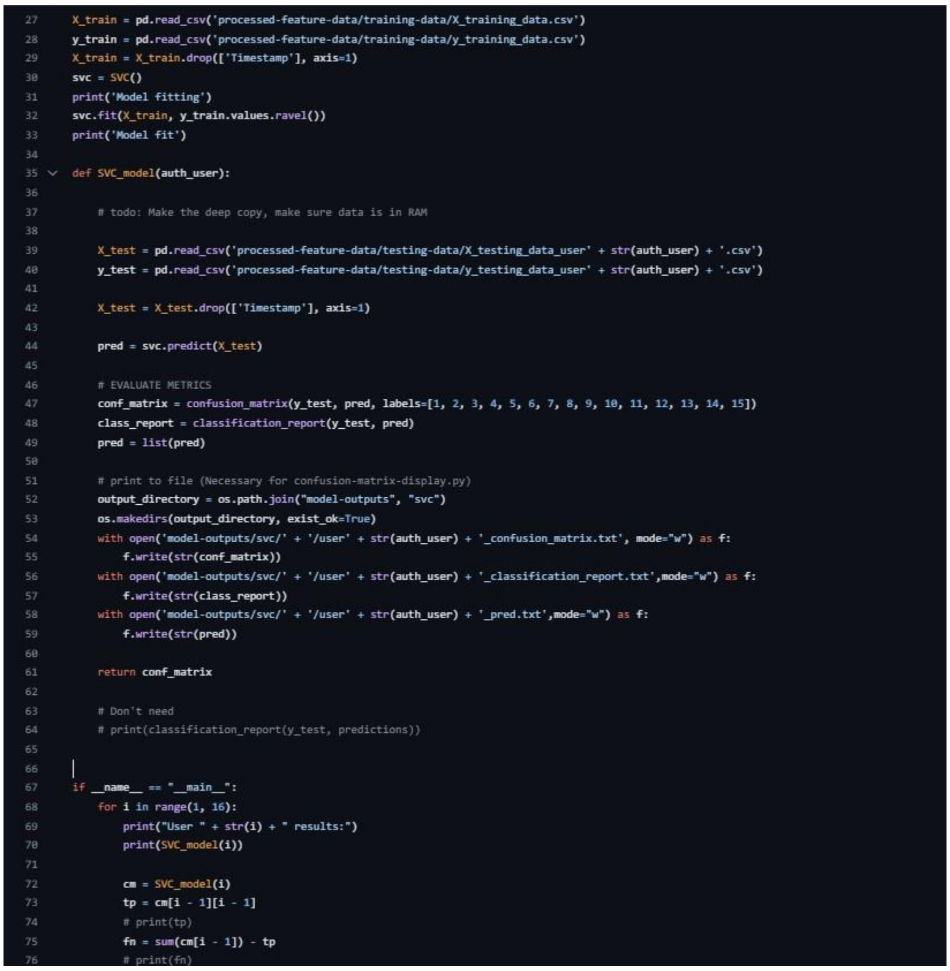


```python
X_train = pd.read_csv('processed-feature-data/training-data/X_training_data.csv')
y_train = pd.read_csv('processed-feature-data/training-data/y_training_data.csv')

# param_grid = {'max_depth':[7, 9, 11, 13], 'max_features':['sqrt','log2'],
#               'min_samples_leaf':[2,3,4,5], 'min_samples_split':[7,8,9,10]}
# print('param grid set.')
#
# grid = GridSearchCV(RandomForestClassifier(), param_grid, verbose=2)
# print('Fitting...')
# grid.fit(X_train,y_train.values.ravel())
# print("Done.")
# print('Grid search complete')

# Found dropping these metrics to be beneficial
X_train = X_train.drop(['Timestamp'], axis=1)

rfc = RandomForestClassifier(n_estimators=300, max_depth=7, min_samples_leaf=3)
print('fitting model')
rfc.fit(X_train, y_train.values.ravel())
print('\nModel fitted.\n')

# todo: Add max_features parameter once you figure out how it works

# todo: Also add max_features implementation


def random_forest_model(auth_user, max_depth=7, min_samples_leaf=3):
    X_test = pd.read_csv('processed-feature-data/testing-data/X_testing_data_user' + str(auth_user) + '.csv')
    y_test = pd.read_csv('processed-feature-data/testing-data/y_testing_data_user' + str(auth_user) + '.csv')

    X_test = X_test.drop(['Timestamp'], axis=1)

    pred = rfc.predict(X_test)

    # EVALUATE METRICS
    conf_matrix = confusion_matrix(y_test, pred, labels=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
    class_report = classification_report(y_test, pred)
    pred = list(pred)

    # print to file (Necessary for confusion-matrix-display.py)
    output_directory = os.path.join("model-outputs", "random-forest", 'max-depth-' + str(max_depth))
    os.makedirs(output_directory, exist_ok=True)
    with open('model-outputs/random-forest/max-depth-' + str(max_depth) + '/user' + str(auth_user) + '_confusion_matrix.txt', mode="w") as f:
        f.write(str(conf_matrix))
    with open('model-outputs/random-forest/max-depth-' + str(max_depth) + '/user' + str(auth_user) + '_classification_report.txt',mode="w") as f:
        f.write(str(class_report))
    with open('model-outputs/random-forest/max-depth-' + str(max_depth) + '/user' + str(auth_user) + '_pred.txt', mode="w") as f:
        f.write(str(pred))

    return conf_matrix
```

Figure 5 – Random Forest Python code

4.1.3. Support Vector Classifier

The SVC works by using the training data to find the most optimal "support vector" to split the data to categorize one class versus another. It then uses the same vector to make classifications with the test data. We used scikit-learn's SVC library and its methods to deploy our own model in Python. It's worth noting that this algorithm was extremely time consuming to execute, especially when initially fitting the model. Some possible hyperparameters that could've been changed were the C and gamma values, but no hyperparameter tuning was necessary with our data and model.



```python
X_train = pd.read_csv('processed-feature-data/training-data/X_training_data.csv')
y_train = pd.read_csv('processed-feature-data/training-data/y_training_data.csv')
X_train = X_train.drop(['Timestamp'], axis=1)
svc = SVC()
print('Model fitting')
svc.fit(X_train, y_train.values.ravel())
print('Model fit')

def SVC_model(auth_user):

    # todo: Make the deep copy, make sure data is in RAM

    X_test = pd.read_csv('processed-feature-data/testing-data/X_testing_data_user' + str(auth_user) + '.csv')
    y_test = pd.read_csv('processed-feature-data/testing-data/y_testing_data_user' + str(auth_user) + '.csv')

    X_test = X_test.drop(['Timestamp'], axis=1)

    pred = svc.predict(X_test)

    # EVALUATE METRICS
    conf_matrix = confusion_matrix(y_test, pred, labels=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
    class_report = classification_report(y_test, pred)
    pred = list(pred)

    # print to file (Necessary for confusion-matrix-display.py)
    output_directory = os.path.join("model-outputs", "svc")
    os.makedirs(output_directory, exist_ok=True)
    with open('model-outputs/svc/' + '/user' + str(auth_user) + '_confusion_matrix.txt', mode="w") as f:
        f.write(str(conf_matrix))
    with open('model-outputs/svc/' + '/user' + str(auth_user) + '_classification_report.txt',mode="w") as f:
        f.write(str(class_report))
    with open('model-outputs/svc/' + '/user' + str(auth_user) + '_pred.txt',mode="w") as f:
        f.write(str(pred))

    return conf_matrix

    # Don't need
    # print(classification_report(y_test, predictions))


if __name__ == "__main__":
    for i in range(1, 16):
        print("User " + str(i) + " results:")
        print(SVC_model(i))

        cm = SVC_model(i)
        tp = cm[i - 1][i - 1]
        # print(tp)
        fn = sum(cm[i - 1]) - tp
        # print(fn)
```

Figure 6 – SVC Python code

5. RESULTS

Most of the evaluations that we made came from the results of true positives, false positives, true negatives, and false negatives. These can be seen below in Figure 4. In the context of this study, a true positive (TP) is defined as when an authentic user is classified as themself, a true negative (TN) is when an impostor is correctly classified, a false positive (FP) is when an impostor is classified as the authentic user, and a false negative (FN) is when an authentic user is classified as an impostor.

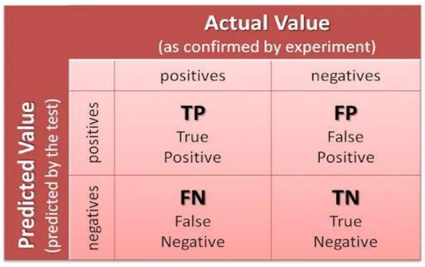


Figure 7 – 2x2 matrix of how TP, FP, FN, and TN are calculated. Image provided by https://medium.com/@awabmohammedomer/confusion-matrix-b504b8f8e1d1

The evaluating our models, we calculated accuracy, precision, recall, f1-score, and area under the curve (AUC). These equations can be seen in Table 2.

Table 2 - Formulas for evaluation metrics

Evaluation Metric	**Formula**
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1	$\frac{2*precision*recall}{precision+recall}$
Accuracy	$\frac{TP+TN}{TP+FN+TN+FP}$

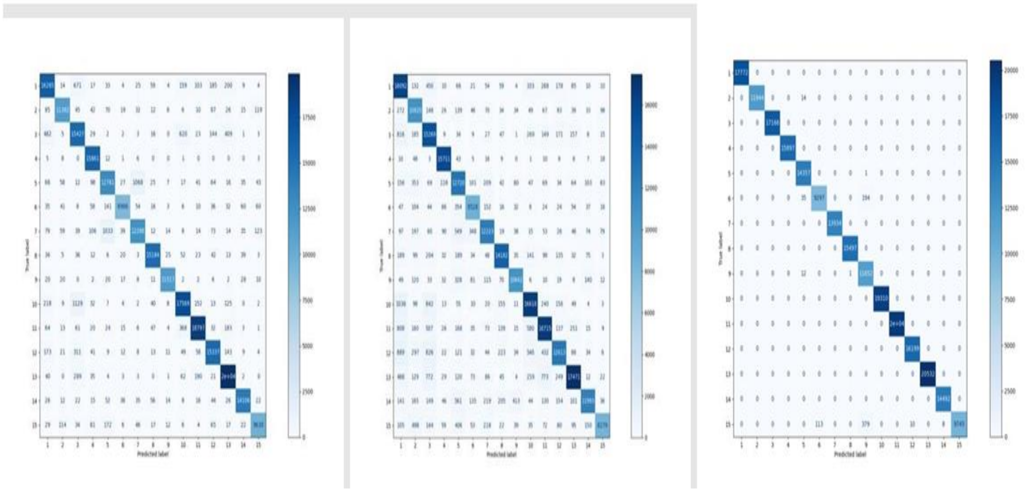


Figure 8 – Confusion matrices for k-NN, SVC, and RF models respectively

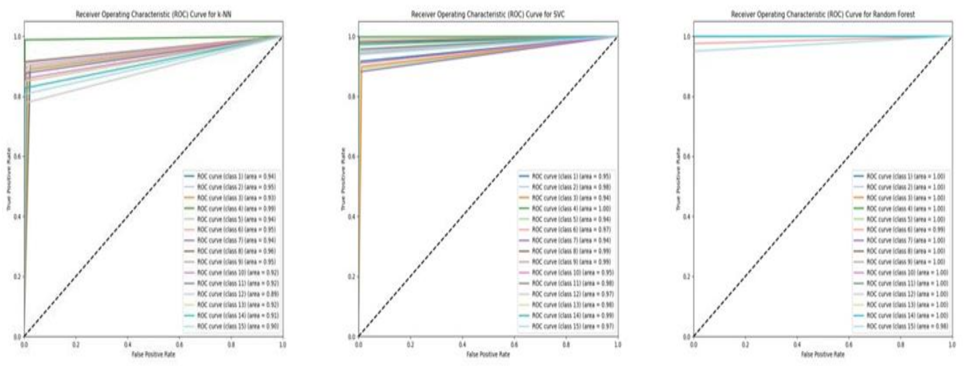


Figure 9 – ROC curves for k-NN, SVC, and RF models respectively

Table 3 – Accuracy test for KNN, SVC, and RF respectively.

User	**k-NN**	**SVC**	**RF**
1	70%	85%	100%
5	73%	80%	100%
7	80%	81%	100%
8	86%	96%	100%
15	78%	90%	95%

We can see from the results for all the models in Figures 5,6 and Table 3 that KNN produced above average results, SVC produced exceptional results, and RF produced results that were too good, and likely a product of overfitting: the model learning from the noise of the training data very well but doing poorly with unseen data. This is an assumption we have made based off comparing the near perfect results we obtained to the results of previous similar research. This assumption could be evaluated through plotting loss and accuracy against various tunings of the model and identifying where our model's test results stray from validating results. Many avenues were taken to combat this overfitting. After pairing multiple sets of hyperparameters and manipulating the data even further with little to no improvement in results, we concluded that the random forest model was not the right fit for our data. Due to this, we will be disregarding the results from the random forest model.

Table 4 - Model evaluation results

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**	**AUC**
k-NN	0.78	0.88	0.88	0.88	0.934
SVC	0.9	0.94	0.94	0.95	~0.9693
~~**RF**~~	~~0.99~~	~~1.00~~	~~0.99~~	~~1.00~~	~~0.998~~

The scoring criteria seen in Table 4 are defined as the following:

- Accuracy: The rate of correct classifications out of all classifications, either authentic or impostor.
- Precision: The rate at which the authentic user defined as themselves.
- Recall: Out of all instances of the correct user, the rate that the model was right in classifications.
- F1-score: The harmonic mean of precision and recall.
- AUC: Generated by the area under the receiver operating characteristic curve. Tests results of the model at different thresholds of sensitivity for true positives and false positives.

6. DISCUSSION AND ANALYSIS

Table 5 - A contrastive evaluation of the models we used and other models from previous research papers.

Paper	Method	Results
[1]	Siamese RNN	EER = 13%, Accuracy = 87%
[2]	Multilayer Perceptron (MLP)	FRR = 2.55%, FAR = 6.94%, Accuracy = 95.96%
[3]	SVC and K-Means	Accuracy = 93.7%, Recall = 86%
[16]	SVC	Accuracy = 97.7%
[8]	Random Forest	Accuracy = 97%, EER = 3.49%
Our Research	KNN and SVC	AUC = 93.4%-97%, Precision = 88%95%, F1-Score = 88%-94%, Accuracy = 78%-90%, FRR = 6% (SVC)

Table 5 provides a comparative analysis of the performance of various machine learning methods used in different research papers and our current study. The methods include Siamese Recurrent Neural Networks (RNN), Multilayer Perceptron (MLP), Support Vector Machines (SVC) and KMeans, Random Forest, and K-Nearest Neighbors (KNN) and Support Vector Classifier (SVC). The performance metrics are accuracy, error rate, and false acceptance rate. The table shows that our most robust model was SVC, which achieved an average accuracy of approximately 90%. This indicates that SVC can effectively distinguish users based on their touch dynamics while playing Minecraft. The table also shows that the other methods, such as RNN, MLP, SVC and K-Means, and Random Forest, also achieved high accuracy rates, ranging from 86% to 97.7%. These results suggest that touch dynamics are a reliable source of behavioural biometrics for continuous authentication. However, the table also reveals some limitations of the methods, such as the high error rate of Siamese RNN (13%) and the high false acceptance rate of Multilayer Perceptron (6.94%). These limitations imply that some methods may be more prone to misclassify users or accept impostors, which can compromise the security of authentication systems. Therefore, further studies are needed to improve the performance and robustness of the methods.

7. LIMITATIONS

The main limitation of this study is the small size and diversity of the dataset. We only collected data from 15 users, who were all students from the same university. This may limit the generalizability of our results to other populations and contexts. Moreover, we only used one type of device (Samsung Tablet) and one type of application (Minecraft) to capture the touch dynamics of the users. Different devices and applications may have different effects on the user's behaviour and performance. Therefore, future work should aim to collect more data from a larger and more diverse sample of users, using different devices and applications, to validate and improve our approach.

8. CONCLUSION

Our research aimed to further understanding in the field of continuous authentication using behavioural biometrics with the goal of expanding authentication systems past the entry point. Understanding and development of increasingly secure authentication systems is crucial as the quantity of personal data being stored on personal devices continues to increase. The results of our research, along with the contribution of a novel dataset add to the foundation of research in this field by demonstrating the possibility of using touch-screen behavioural biometrics for continuous authentication without clustering data points into segments. The results of our research, along with the research done by other teams suggest that some machine learning models are more fit for this use case than others. As the quantity of personal data stored on touch screen devices continues to increase, the implications of our research and a touchscreen based continuous authentication system become more evident. A system reliable enough to accurately determine the identity of users while they interact with a device would have significant contributions to the field of cyber security and personal data protection.

REFERENCES

[1] Acien, A.; Morales, A.; Vera-Rodriguez, R.; Fierrez, J. Smartphone sensors for modeling humancomputer interaction: General outlook and research datasets for user authentication. In Proceedings of the 2020 IEEE 44th Annual Computers, Software, and Applications Conference, Madrid, Spain, 13 July 2020. [CrossRef]

[2] Liang, X.; Zou, F.; Li, L.; Yi, P. Mobile terminal identity authentication system based on behavioral characteristics. Int. J. Distrib. Sens. Netw. 2020, 16, 1550147719899371. [CrossRef]

[3] Torres, J.; Santos, S.; Alepis, E.; Patsakis, C. Behavioral Biometric Authentication in Android Unlock Patterns through Machine Learning. In Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP), Prague, Czech Republic, 23–25 February 2019. [CrossRef]

[4] Shi, D.; Tao, D.; Wang, J.; Yao, M.; Wang, Z.; Chen, H.; Helal, S. Fine-grained and context-aware behavioral biometrics for pattern lock on smartphones. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2021, 5, 1–30. [CrossRef]

[5] J. Cybersecur. Priv. 2023, 3 256 22. Fierrez, J.; Pozo, A.; Martinez-Diaz, M.; Galbally, J.; Morales, A. Benchmarking touchscreen biometrics for mobile authentication. IEEE Trans. Inf. Forensics Secur. 2018, 13, 2720–2733. [CrossRef]

[6] Lee, J.; Park, S.; Kim, Y.G.; Lee, E.K.; Jo, J. Advanced Authentication Method by Geometric Data Analysis Based on User Behavior and Biometrics for IoT Device with Touchscreen. Electronics 2021, 10, 2583. [CrossRef]

[7] Antal, M.; Szabó, L.Z. Biometric authentication based on touchscreen swipe patterns. Procedia Technol. 2016, 22, 862–869. [CrossRef]

[8] Zhang, X.; Zhang, P.; Hu, H. Multimodal continuous user authentication on mobile devices via interaction patterns. Wirel. Commun. Mob. Comput. 2021, 2021, 5677978. [CrossRef]

[9] Samet, S.; Ishraque, M.T.; Ghadamyari, M.; Kakadiya, K.; Mistry, Y.; Nakkabi, Y. TouchMetric: A machine learning based continuous authentication feature testing mobile application. Int. J. Inf. Technol. 2019, 11, 625–631. [CrossRef]

[10] Pelto, Brendan, Mounika Vanamala, and Rushit Dave. "Your Identity is Your Behavior--Continuous User Authentication based on Machine Learning and Touch Dynamics." *arXiv preprint arXiv:2305.09482* (2023).

[11] Z. DeRidder *et al.*, "Continuous User Authentication Using Machine Learning and Multi-finger Mobile Touch Dynamics with a Novel Dataset," *2022 9th International Conference on Soft Computing & Machine Intelligence (ISCMI),* Toronto, ON, Canada, 2022, pp. 42-46, doi: 10.1109/ISCMI56532.2022.10068450.

[12] Leyfer, K.; Spivak, A. Continuous user authentication by the classification method based on the dynamic touchscreen biometrics. In Proceedings of the 2019 24th Conference of Open Innovations Association (FRUCT), Moscow, Russia, 8 April 2019. [CrossRef]

[13] Li, W.; Meng, W.; Furnell, S. Exploring touch-based behavioral authentication on smartphone email applications in IoT-enabled smart cities. Pattern Recognit. Lett. 2021, 144, 35–41. [CrossRef]

[14] Buriro, A.; Gupta, S.; Yautsiukhin, A.; Crispo, B. Risk-driven behavioral biometric-based one-shotcum-continuous user authentication scheme. J. Signal Process. Syst. 2021, 93, 989–1006. [CrossRef]

[15] Levi, M.; Hazan, I.; Agmon, N.; Eden, S. Behavioral embedding for continuous user verification in global settings. Comput. Secur. 2022, 119, 102716. [CrossRef]

[16] Miyamoto, N.; Shibata, C.; Kinoshita, T. Authentication by Touch Operation on Smartphone with Support Vector Machine. Int. J. Inf. Secur. Res 2017, 7, 725–733. [CrossRef]