ITI105 Final Report

**User Authentication Using Classical Machine Learning:**

**Leveraging Key Typing Dynamics Behavior**

*Allen Lee and Jacob Abraham*

*25-Aug-2023*

# Problem statement and solution

Computing devices, including mobile phones, use various biometric authentication methods like fingerprints or facial features to identify users. However, these methods rely on specific hardware, which can increase the overall cost. An economical alternative is to authenticate users based on their behaviour, such as typing dynamics.

Keystroke dynamics, also known as keystroke biometrics, pertain to the comprehensive timing data that precisely records the moment each key is pressed and released as an individual types on a computer keyboard. These dynamics offer valuable insights that aid in user authentication. Through capturing the intervals between key presses, key hold durations, and the periods between key releases and next key presses, significant user insights can be derived. When the user logs in again, companies can compare their present typing pattern with their past patterns, allowing for authentication to distinguish legitimate users from potential fraudulent ones.

Conventional machine learning will be applied to generate models that can infer keystroke dynamics and then authenticate users.

# ML Problem Articulation and Formulation

* **Problem statement**: We want to create a machine learning model that accurately identifies users based on their typing dynamics, achieving a high level of authentication performance and provide a cost-effective alternative to biometric authentication methods for computing devices.
* **Current solution**: The current solution involves using biometric authentication methods such as fingerprints or facial features, which require specialized hardware and can be costly.
* **Ideal outcome**: The ideal outcome would be a reliable and cost-effective user authentication system that eliminates the need for specialized hardware, making it accessible and affordable for a wide range of computing devices.
* **Success metrics**: High authentication accuracy, Cost-effectiveness, Consistent and stable performance over time, Robustness against impersonation, and Positive user acceptance.
* **Failure metrics**: False accept rate, False reject rate, Inconsistency in user authentication, Vulnerability to impersonation, and low user acceptance and satisfaction levels.
* **Input to the model**: Keystroke timings (21 features) obtained from 21 users, typing the password ‘.tie5Roanl’.
* **ML model:** Our problem is best framed as a Supervised **Multi-class Classification task**, which predicts whether the user’s typing dynamics match the authorized user profile or not.
* **Output of the ML model**: The output of the machine learning model would be a multi-class decision indicating whether the user's typing dynamics match the authorized user profile or not.
* When a new request for authentication is presented to the model, if the user’s keystroke data matches to the trained data, the user will be authenticated, else the user will not be authenticated.
* The outcome will be displayed either as “**(Subject ID)”** or “**Unidentified Subject**”, if the prediction probability is below a threshold.
* **Using the Output**: Access control, Security enhancement, User verification, and Fraud detection.
* **Hypotheses, and assumptions**: Typing dynamics are sufficiently distinct and consistent among individuals, and that there are enough data samples available to train and evaluate the machine learning model effectively. Individuals have unique typing patterns that can be captured and used for user authentication, and that these patterns remain consistent over time.

# Final Data Preparation & Feature Engineering

## 3.1 Definitions of Input Data

Critical information that is captured during typing a password are:

* 1. **Hold duration (H)**: Time from when a key was pressed to when it was released.
  2. **Up-to-Down (UD) duration**: Time from when ‘A’ *key* was released to when ‘B’ *key* was pressed, as shown below.
  3. **Down-to-Down (DD) duration**: Time from when ‘A’ *key* was pressed to when ‘B’ *key* was pressed, as shown below.

A black screen with a black rectangle and arrows

Description automatically generated

*Figure-1. Critical keystroke timings*

## 3.2 Description of Dataset

* Benchmark dataset was downloaded from Carnegie Melon University http://www.cs.cmu.edu/~keystroke/.
* Data consist of keystroke-timing information from 51 subjects (typists), each typing same 10-character password “.tie5Roanl” 400 times (in 8 sessions, with 50 repetitions per session).
* One additional user data (around 400 keystroke timing samples from Allen) was collected.
* **Final dataset**: Created a hybrid dataset by merging the first 10 users’ samples and Allen’s 414 keystroke samples.
* Total number of samples is 4414.
* **Final features (and their data types):**

A screenshot of a computer

Description automatically generated

*Note:*

*H.period: during the period “.” Key was held.*

*DD.period.t: Time from when period “.” was pressed to when “t” key was pressed.*

*UD.period.t: Time from when period “.” was released to when “t” key was pressed.*



## 3.3 Data Analysis and Visualisation with Final Dataset

* Total number of samples is 4414.
* Total number of original columns is 33.

A computer screen shot of a computer code

Description automatically generated

* Target is column “subject”, and its data type is “object”. Data type in other columns are “float”.
* Columns “sessionIndex” and “rep” are not used as it will not affect model training and testing.
* There are no missing values and “NaN” (refer to section 2.1). Hence, minimal data cleaning / imputation was needed.
* Columns of “Hold” duration shows low standard deviation compared to their corresponding columns of “Up-to-Down”.
* Values in all columns of “Down-to-Down” are the sum of their corresponding values in columns of “Hold” and “Up-to-Down”. Columns of “Up-to-Down” shows strong correlation with column “Down-to-Down”. Calculated correlation coefficients for these pairs of ‘Down-to-Down' and ‘Up-to-Down' for each character keypress are above 0.9 as shown below.



* Hence all columns prefixed with “DD”, are to be dropped to reduce the effects on model training due to correlated features. This reduces the number of features from 31 to 21.
* Final features:

A screenshot of a computer code

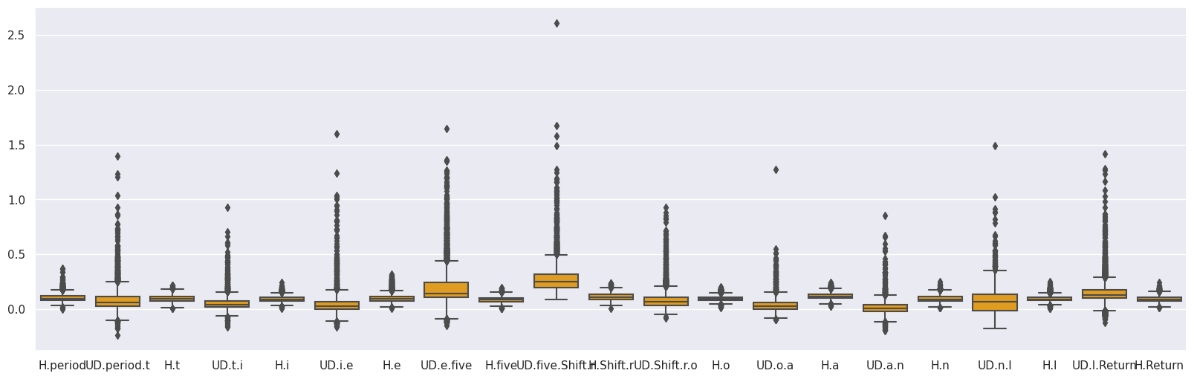
Description automatically generated

* It is also noted that “Up-to-Down” values can be negative. The range of “UD.period.t” is from -0.24 to 12.45. This implies that the next key in sequence was pressed before the release of the previous key.

A screen shot of a screen

Description automatically generated

* All the “Up-to-Down” features have outliers. Will run experiments to know their effects on the model accuracy.



* All numerical features of the dataset were scaled to ensure that the features have comparable magnitudes. Standard scaler was applied.

## 3.4 Feature Engineering

* Feature importance study was carried out using Random Forest Classifier and the ‘importance’ values obtained were listed as below.

A graph with blue and white bars

Description automatically generatedA table of numbers with black text

Description automatically generated

* The feature “UD.n.l” contributed 13.5 % to the feature’s importance.
* Generating extra attributes is achievable through the calculation of the ratio between the "Hold" time and the variance in total time to complete typing of all characters in the password. But due to the time limitation, experiments with engineered features were not carried out to see their effects on model accuracies.

## 3.5 Outliers

* Outliers were also observed with analysis and visualization, as show in Figure below.

A graph of different sizes and shapes

Description automatically generated with medium confidence

* Performance of Random Forest Classier with and without outliers was found to be the same.

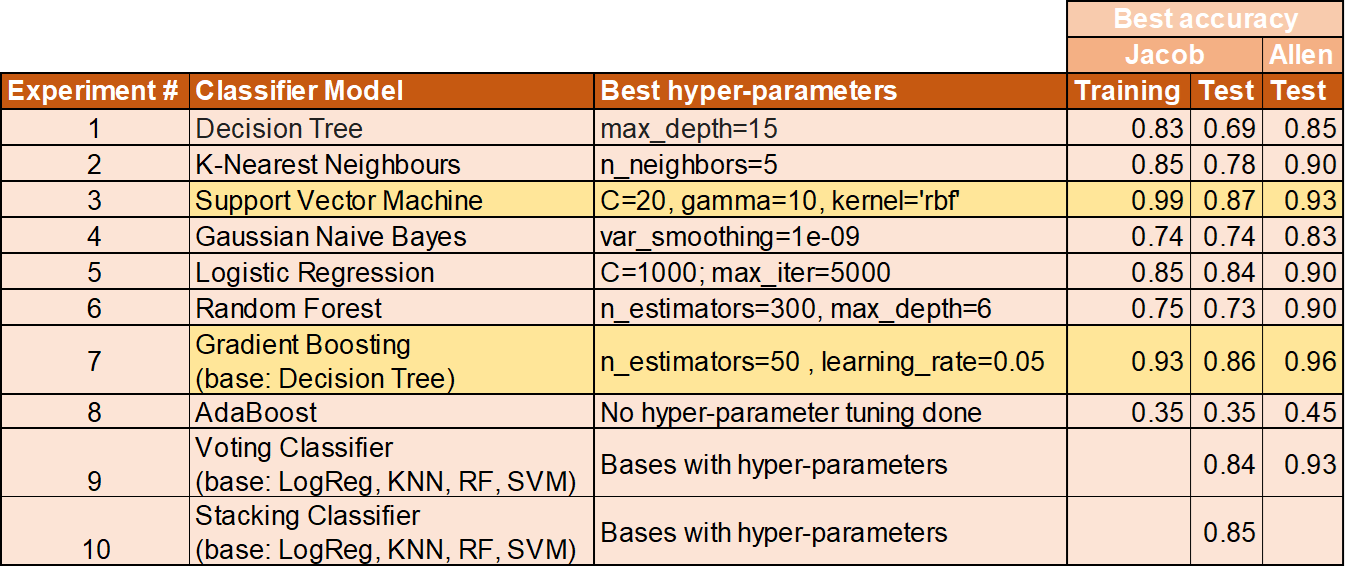
# Modelling and Experiments

This section focuses on the hypothesis, selection of the best model that will be used to test the hypothesis, error analysis, observations, and performance tuning.

An initial evaluation runs on all the general models were carried out by both Jacob and Allen to screen out suitable models that can give the best results for such keystroke dynamic data. The initial screenings were done using the full dataset from the Carnegie Melon University study while we were collecting our own data in mean time. Some minimal hyperparameters tuning were done for each model. Single model and ensemble models were covered.

## 4.1 Summary Results from the Initial Screening with Benchmark Dataset

* The summary of experiments with base and ensemble learners is provided below.



* The best model accuracies obtained so far are with SVM and Gradient Boosting (with the default base classifier Decision Tree) ensemble learning. This is consistently shown in the accuracy scores from both Jacob and Allen’s studies.
* In the next section, further fine-tuning and analysis will be done on SVM and Gradient Boosting models using our captured data with a subset of the benchmark dataset.
* Conducting experiments is a time-consuming process due to the significant amount of machine time needed for hyperparameter tuning. MLFlow is used by Allen to log the experiment parameters and results (see the exported csv files under ‘Experiment Logs’ directory), while Jacob logged the results in the Python codes and charted them (see .jpynb file under ‘Final\_Submission’ directory).

## 4.2 Evaluation on SVM and Gradient Boosting Models with Final Dataset

We proceeded to narrow the selection of the final model between SVM and Gradient Boost ensemble models. A subset of the CMU dataset from the first 10 subjects and new training data from Allen’s own keystroke dynamic were captured and used for this round of evaluation.

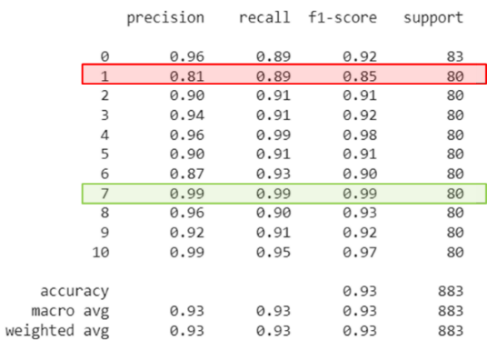
### 4.2.1 SVM Hyperparameter Tuning and Evaluation

There were 2 kernels being evaluation for Support Vector Machine classifier. They were Radial Basis Function (RBF) kernel and Polynomial kernel. The evaluation results were calculated from their mean cross-validation score on the training dataset, as well as the prediction accuracy scores on the test dataset from the training/testing split.

For ‘RBF’ kernel, grid search runs were done on the regularisation parameter C and gamma value. A summary of the results is as follows:

**SVM with ‘RBF’ kernel**

* Hyper-parameters Grid Search range
  + C: [0.001, 0.01, 0.1, 1, 10, 100]
  + Gamma: [0.001, 0.01, 0.1, 1, 10, 100]
* Best parameters: {'svm\_\_C': **10**, 'svm\_\_gamma': **0.01**}
* Mean cross validation accuracy score: 0.94
* Test set score: 0.93
* Classification report and analysis

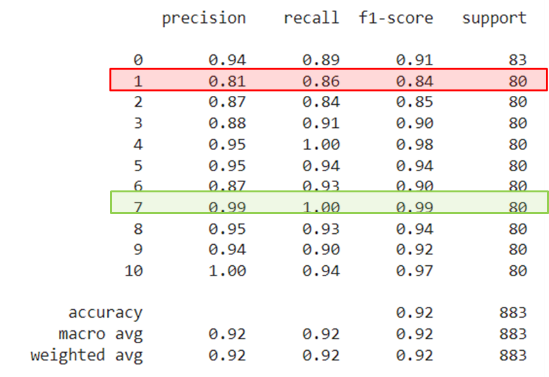


* + - Subject-s002 has the worst metrics among 11 subjects.
    - Subject-s010 has the best metrics among 11 subjects.

For ‘Polynomial’ kernel, grid search runs were done on the regularisation parameter C and polynomial degree values. A summary of the results is as follows:

**SVM with ‘Poly’ kernel**

* Hyper-parameters Grid Search range
  + C: [0.001, 0.01, 0.1, 1, 10, 100]
  + degree: [1, 2, 3, 4, 5]
* Best parameters: {'svm\_\_C': **10**, 'svm\_\_degree': **3**}
* Mean cross validation accuracy score: 0.93
* Test set score: 0.93
* Classification report and analysis



* + - Subject-s002 has the worst metrics among 11 subjects.
    - Subject-s010 has the best metrics among 11 subjects.

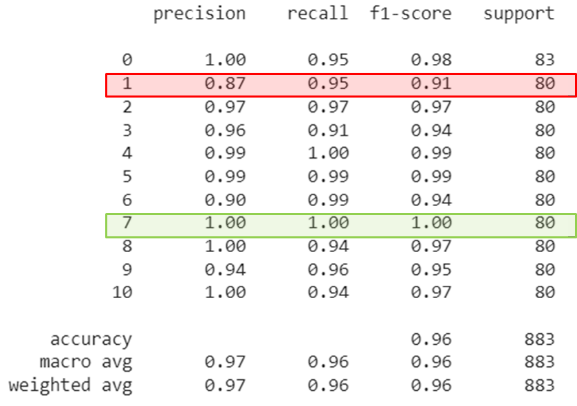
The results for SVM classifiers using ‘RBF’ and ‘Polynomial’ kernel functions were quite close.

### 4.2.2 Gradient Boosting Hyperparameter Tuning and Evaluation

The gradient boosting model is an ensemble model. Grid search runs were done on the learning rate, estimator count and fraction of sample used for fitting base estimators. Similarly, the mean cross validation score on the training data and the prediction score on the testing data from the train/test data split were used as evaluation metrics. A summary of the results is as follows:

**Gradient Boosting**

* Hyper-parameters Grid Search range
  + learning\_rate: [0.01, 0.1, 1, 10]
  + n\_estimators: [200, 300]
  + subsample: [0.8, 0.9, 1.0]
* Best parameters: {**learning\_rate**=**0.1**, **n\_estimators**=**300**, **subsample**=**0.9**}
* Best cross validation accuracy: 0.97
* Test set score: 0.97
* Classification report and analysis



* + - Subject-s002 has the worst metrics among 11 subjects.
    - Subject-s010 has the best metrics among 11 subjects.

## 4.3 Final Model selection

* Predictions were consistent among the 3 Models (SVM-RBF, SVM-Poly, and GB) with respect to subjects’ prediction metrics.

A table with numbers and symbols

Description automatically generated

* Gradient boosting gives the best metrics among the three models and hence selected as final model.
* This proves the hypothesis that keystroke dynamics can be successfully applied for user authentication due to this high accuracy.

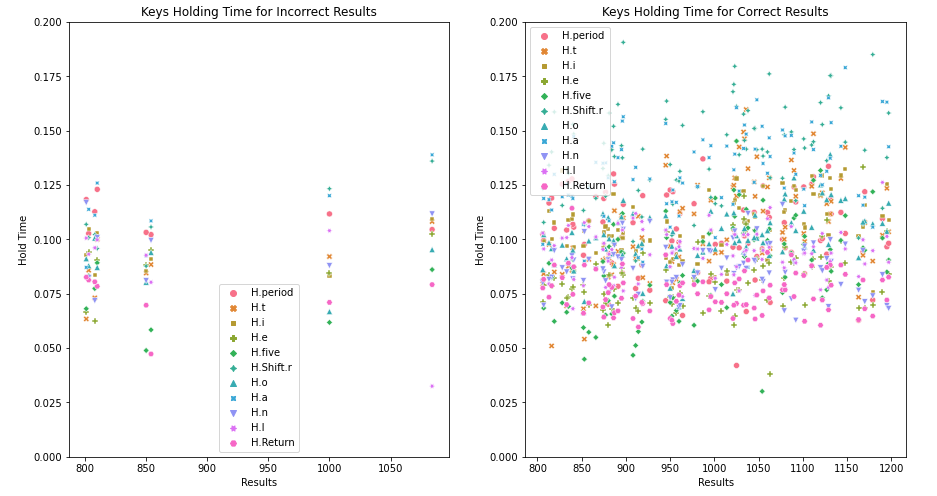
## 4.4 Error Analysis

* Confusion matrix plot on the test data prediction from Gradient Boosting ensemble indicates that 6 samples from the subject ‘s004’ were classified wrongly as ‘s002’. This represents **7.5%** of the ‘s004’ samples, which is relatively higher than the rest.

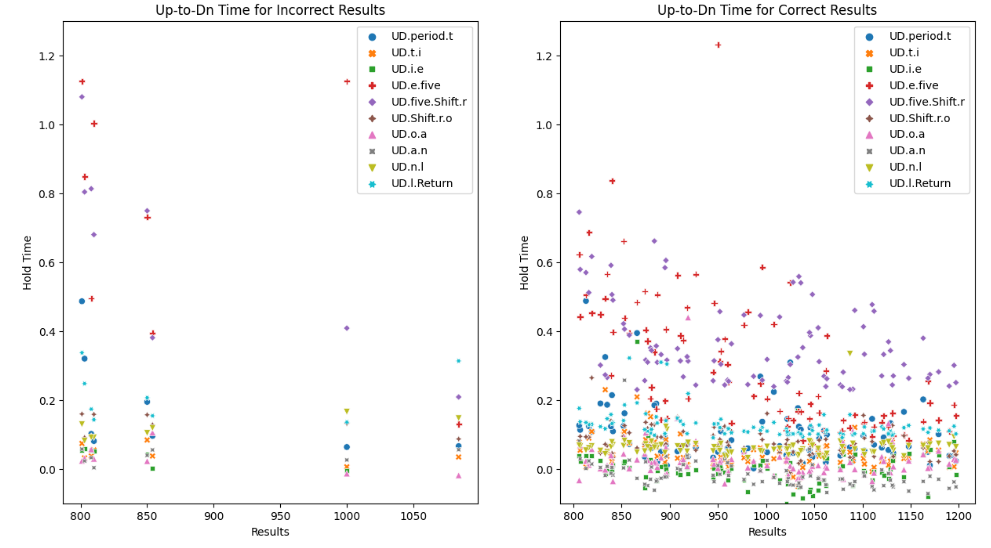
A graph of numbers and a number

Description automatically generated with medium confidence

* We analysed the key press hold times for the samples that were incorrectly classified against that of samples that were correctly classified for the ‘s004’ subject on the same y-scale.



* It appears that the key hold duration is generally **longer** **for those correctly classified** samples **than** those which are **incorrectly classified** by the model, especially for some keys like **'H.Shift.r**' (Shift-R key press). This suggests that the underlying characteristics of the key stroke dynamics in the incorrectly classified samples were different from that of the classified samples.
* Next, we analysed the key transfer duration from Up-to-Down between the key sequence.



* In the correctly classified chart, we could see that there are many samples with negative key travel timing, especially for ‘UD.n.l’ and ‘UD.l.Return’ parameters. This suggests that the subject has pressed the next key before releasing the previous key in sequence. However, in the incorrectly classified samples, these values were mostly in the positive region.
* A few of the transfer key durations for the incorrectly classified samples were well above 0.8s, compared to the correctly samples where only 2 samples (‘UD.e.five’) were observed to be above 0.8s.
* Similarly, this also indicated that the underlying keystroke dynamics of these incorrectly classified samples were not like the rest of the samples from the same subject.
* In addition, the incorrectly classified samples tend to cluster around 4 sampling sessions as can be seen in the visualization. This might be due to some abnormal keypresses from the subject during those sessions or day when data was captured.
* There might be various factors contributing to this abnormality, including the use of different keyboards, subject’s mood for the day, or even incorrectly labelled data, etc.
* Based on this analysis, we were confident of the performance of the trained model as long as data has consistent characteristics.

# Application Deployment

**Deployment**

The trained model is then serialised into disk via pickle for deployment. It is wrapped into a web REST API service using Flask library(https://flask.palletsprojects.com/). A web application is then developed to provide the front end UI and interface with this web API vis data exchange in the json form.

**Description of the app**

A blue rectangular object with white text

Description automatically generated

**Technology stack and Application Architecture**

A diagram of a computer network

Description automatically generated

Since the user is interacting with the application via a browser, the key stroke dynamics is captured via javascript event triggering mechanism within the browser. It then sends the captured time intervals to the web server via http call. The web server in turn validates the keys entered and then packaged the captured timings into a json list and post to the Python/Flask web via REST API call.

The Python/Flask API module loads the model and calls the prediction function. It then returns the classification results and the prediction probability to the web server in json format.

The web server will then read the returned results and validate that the prediction probability must be above a threshold (initially set at 0.8). If the probability is above the threshold, the web server will authenticate the user as the classified subject. If the probability is below the threshold, it will not authenticate the user and display the result as ‘Unidentified’.

# Conclusion

**Performance summary**

* Gradient Boosting model with high performance metrics (Precision and Recall) has proved the hypothesis that keystroke dynamics can be applied for user authentication as a proof of concept.
* However, many other factors need to be considered for practical implementation. It might not be a practical approach in the real world if each new user needs to type the key sequence 100 or 200 times for the model to be re-trained before the user can be authenticated.
* In addition, a user might not be that consistent with the keystroke every time and when the keyboard or environment changes, the response may differ.
* As the original users are not available for actual inference, user satisfaction, acceptance rate and other success / failure metrics could not be measured.

**Next steps**

* Hardware and time constraints in modelling has limited the hybrid dataset sample size to 4414.
* With better hardware, all (20400) the original benchmark dataset samples can be used to model in a reasonable amount of time.

**Lessons learnt**

* ML Formulation
* Data Analysis, Feature selection, Cross validation, base estimators, and ensemble learning.
* ‘MLFlow’ and application deployment.

# Contributions

**Jacob**

* Sourced initial c code for “Key-Sniffer’.
* Data visualization and Model Evaluation with Decision Tree, K-Nearest Neighbors, Support Vector Machine, Gaussian Naive Bayes, Logistic Regression, Random Forest, AdaBoost, Gradient Boosting, Voting and Stacking ensembles.
* Platform: Google CoLab

**Allen**

* Upgrading and fine tuning of “**Key-Sniffer**” program to collect keystroke data.
* Data visualization and Model Evaluation with Decision Tree, K-Nearest Neighbours, Naive Bayes, Logistic Regression, Support Vector Machine, Random Forest, Gradient Boosting, and Voting Classifier
* Experiment logging.
* Error Analysis.
* Model deployment and inference.
* Platform: **Jupyter Notebook**

**GitHub link**

<https://github.com/sokonana/NYP-IT105-Project>

# References

[1] Keystroke Dynamics - Benchmark Data Set: <http://www.cs.cmu.edu/~keystroke/>

[2] KDA on benchmark: <https://www.kaggle.com/code/ashusrivastava/kda-on-benchmark>

[3] Keystroke Dynamics Analysis and Prediction w/ XGB: <https://www.kaggle.com/code/kartik2112/keystroke-dynamics-analysis-and-prediction-w-xgb>

[4] Keystroke Dynamics Analysis and Prediction — Part 1/2 (EDA): <https://towardsdatascience.com/keystroke-dynamics-analysis-and-prediction-part-1-eda-3fe2d25bac04>

# Appendix

## Modelling base learners with benchmark dataset

Experiments were conducted with various base models on the benchmark dataset.

**Decision Tree**

* Hyper-parameters

max\_depth: [5, 10, 15, 20, 30, 40]

* Train and test performance

A graph showing a line

Description automatically generated

**K-Nearest Neighbours**

* Hyper-parameters

n\_neighbors: [1, 2, 3, 4, 5, 6, 7, 8]

* Train and test performance

**A graph with a line and a line

Description automatically generatedA graph with a line

Description automatically generated**

**Naive Bayes**

* Hyper-parameters

'var\_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]

* Train and test performance

Fitting 5 folds for each of 5 candidates, totalling 25 fits

Best\_parameters: {'var\_smoothing': 1e-09}

Best accuracy score: 0.732

**Logistic Regression**

* Hyper-parameters

C: [1e-3, 1e-2, 0.1, 1, 10, 100, 1000, 10000]

* Train and test performance

**A graph with a line

Description automatically generated**

**Support Vector Machine**

* Hyper-parameters

'C' : [0.01, 0.1, 1, 10, 20, 30]

'gamma' : [0.1, 1, 5, 10, 15, 20]

'kernel': ['rbf', 'poly']

best\_parameters: C=20, gamma=10, kernel='rbf'

* Train and test performance

Accuracy on training set: 0.99

Accuracy on test set: 0.87

Experiment runs are logged using “MLFlow” as shown Appendix 9.1

## Ensemble modelling

**Random Forest**

* Hyper-parameters

n\_estimators = [50, 100, 200, 300, 400]

max\_depth = [3, 4, 5, 6]

* Train and test performance

**A chart of different colors

Description automatically generatedA chart of different colors

Description automatically generated**

**AdaBoost**

* Hyper-parameters

estimator=log\_clf, n\_estimators=100

* Train and test performance

Accuracy on training set: 0.35

Accuracy on test set: 0.35

**Gradient Boosting**

* Hyper-parameters

n\_estimators: = [200, 300]

learning\_rate = [0.05, 0.1]

* Train and test performance

**Voting Classifier**

estimators = [('lr',log\_clf),('knn',knn\_clf),('rf',rf\_clf),('svm', svm\_clf)]

* Hyper-parameters

estimators=estimators, voting='soft

* Train and test performance

Mean Accuracy: 0.837

Std Deviation: 0.091

**Stacking Classifier**

estimators = [('lr',log\_clf),('knn',knn\_clf),('rf',rf\_clf),('svm', svm\_clf)]

* Hyper-parameters

estimators=estimators, final\_estimator=DecisionTreeClassifier()

* Train and test performance

Mean Accuracy: 0.848

Std Deviation: 0.072

* Experiment runs are logged using “MLFlow” as shown Appendix 9.2

## MLFlow Log: Single Model Evaluation Experiment

A screenshot of a computer

Description automatically generated

## MLFlow Log: Ensemble Model Evaluation Experiment

A screenshot of a computer

Description automatically generated

Refer to ‘Experiment Logs’ on GitHub.