**CS 271- Final**

**Machine Learning Based User Authentication through Mouse Dynamics**

Phanidhar Sai Sravan Chandana - 017405531

Rashmi Sonth - 017404244

## **Abstract**

Traditional static authentication methods, such as passwords, face increasing vulnerability due to technological advancements and evolving attack strategies. Continuous authentication emerges as a viable solution, wherein users granted account access are continually monitored to ensure ongoing verification against potential impostors with unauthorized access. Mouse dynamics, encompassing a user's mouse movement behavior, proves to be a promising biometric for continuous authentication schemes. Several evaluation scenarios are explored: RNN models including LSTM, BiLSTM and GRU models. A deep dive here the RNN models have been experimented with Embeddings and without embeddings. Additionally SVM and Random forest models have also been tested. The peak average test accuracy of 91.2% was achieved with BiLSTM models implemented with embeddings among the deep learning models.

## **Introduction and Problem statement**

The challenge at hand involves safeguarding a group of users from unauthorized access to their accounts by leveraging the distinctive characteristics of their mouse movement behavior. These users regularly access remote servers through a remote desktop client, with network traffic monitored by a device situated between the client and the remote computer, adhering to the RDP protocol. This monitoring captures the mouse interactions transmitted during the remote sessions.

Periodically, instances arise where user accounts are illicitly accessed and used by unauthorized individuals, posing a security threat. The objective is to devise a method to detect such unauthorized usages by analyzing the mouse movement data recorded by the network monitoring device.

Given the assumption that an individual's mouse movement patterns are unique and akin to a behavioral biometric identifier, the challenge is to develop a model capable of identifying the characteristic patterns of each user. By doing so, the model can discern instances where a user account is being misused by recognizing deviations in the mouse movement data transmitted from the client to the remote server. The goal is to create an effective detection system that can promptly identify instances of stolen accounts by discerning anomalies in mouse behavior.

In essence, the task is to construct a robust model that can learn and differentiate the typical mouse movement patterns of legitimate account owners and subsequently identify instances of account misuse based on deviations from these established patterns. The model is expected to contribute to enhancing security measures and promptly detecting unauthorized access to user accounts.

## **Solution**

### **1. Understanding and Preprocessing of dataset**

DATASET URL: <https://github.com/balabit/Mouse-Dynamics-Challenge>

The Dataset used for the project is the Balbit mouse dynamics challenge. The collection is composed of anonymized and aggregated data related to mouse movements, encompassing the behaviors of 10 unique users. This data gathering occurred over a two-month period, during which participants conducted their regular day-to-day work tasks. The dataset includes a total of 5,500 sessions, with an average of approximately 55 sessions per user per day.

The data consists of a collection of csv files, each file representing session data for a particular user. The columns in the file are as follows -

* record timestamp - The unix timestamp of the mouse event
* client timestamp - The client side timestamp of the recorded mouse event.
* button - The status of the mouse during the event, it takes values like NoButton, Scroll, Left, Right
* state - state of the mouse, has values like Move, Up, Down, Pressed, Released
* x - The x-coordinate of the mouse on the screen
* y - The y-coordinate of the mouse on the screen

We have fetched the user for each session and then attached it as another column in the dataset.

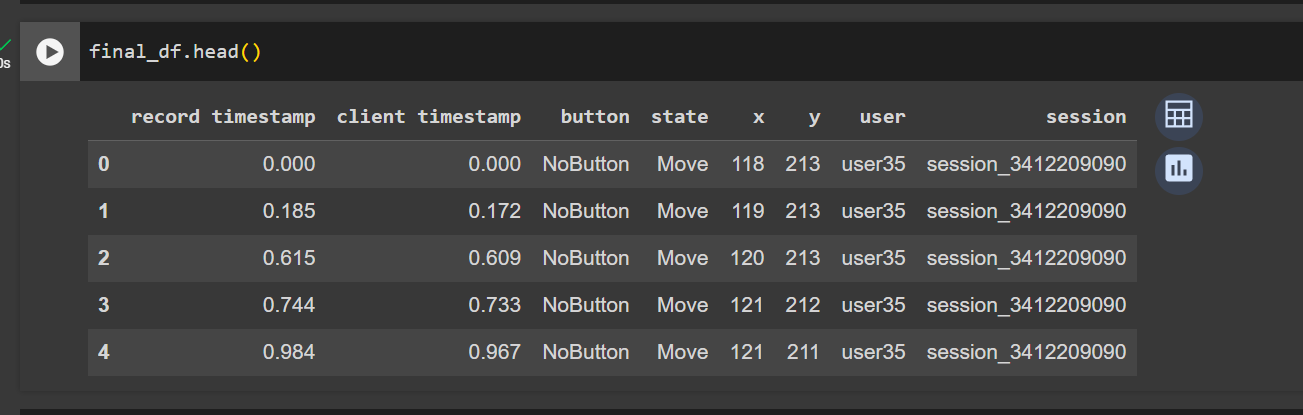


Fig 1.1

We observe that a few columns like button, state user and session are categorical information. Applying label encoder on these columns will convert this data to numerical representation. We have merged our train dataframe and test data frames and then performed encoding, to make sure that a certain value like “user35” is encoded to the same numerical value in both test and train data. And then again split the test data and train in the same way they were before.

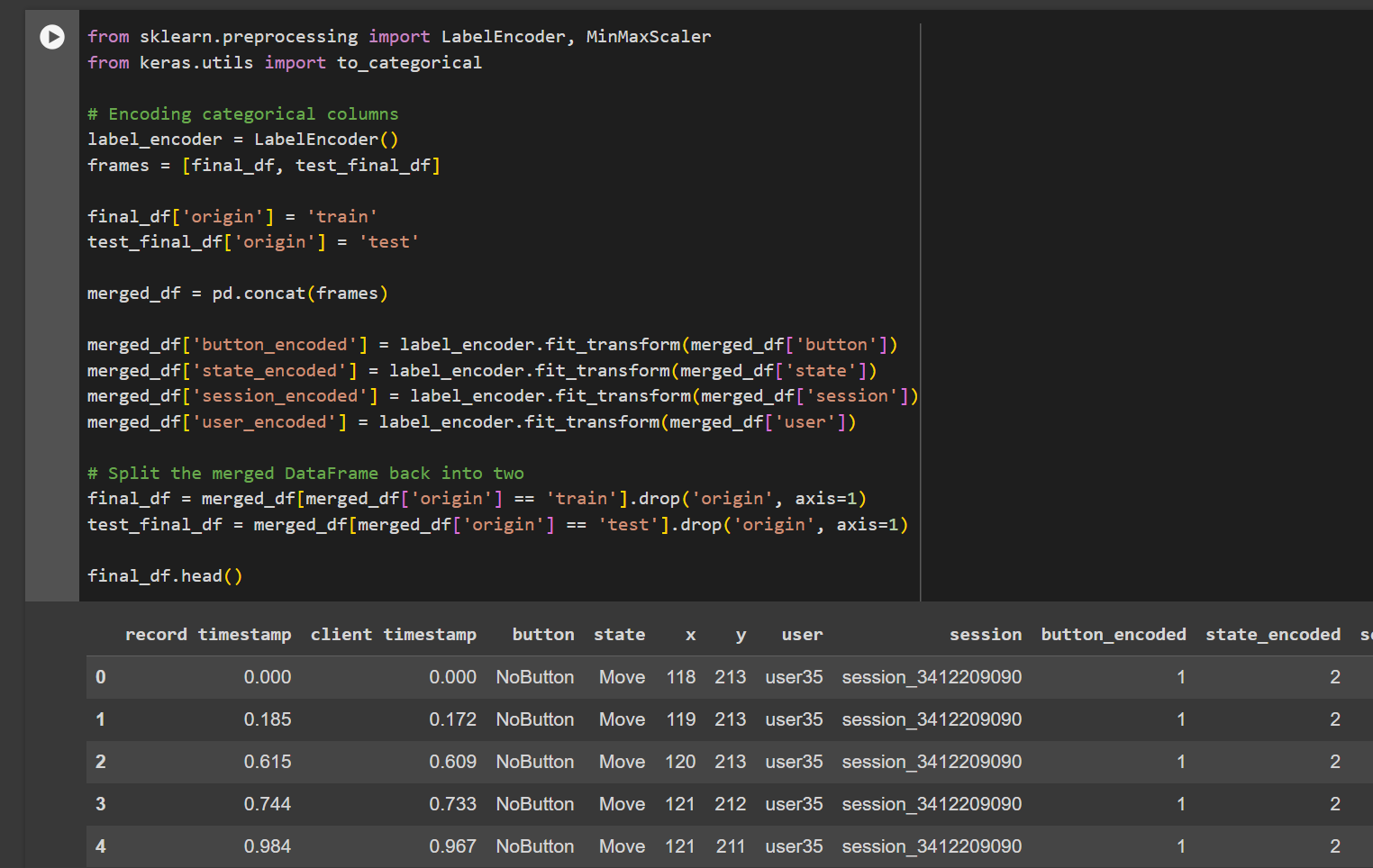


Fig 1.2

And then we dropped the categorical columns so that our new updated data frame looks as shown in fig 1.3

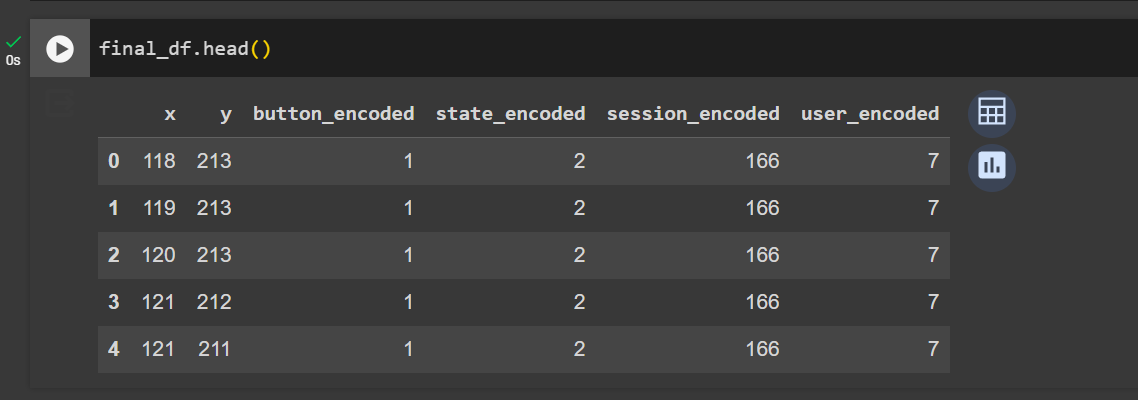


Fig 1.3

## **Methodologies**

### **1. LSTM**

We used LSTM to classify the users for the sessions. We opted for LSTM as the data is sequential in nature, for example a time series, where you have data points x\_t for multiple time steps t=t0...tN.Here, N would be the sequence length of a session. We have found out that the **shortest session in the data has 13456 timesteps** in both test and train data.

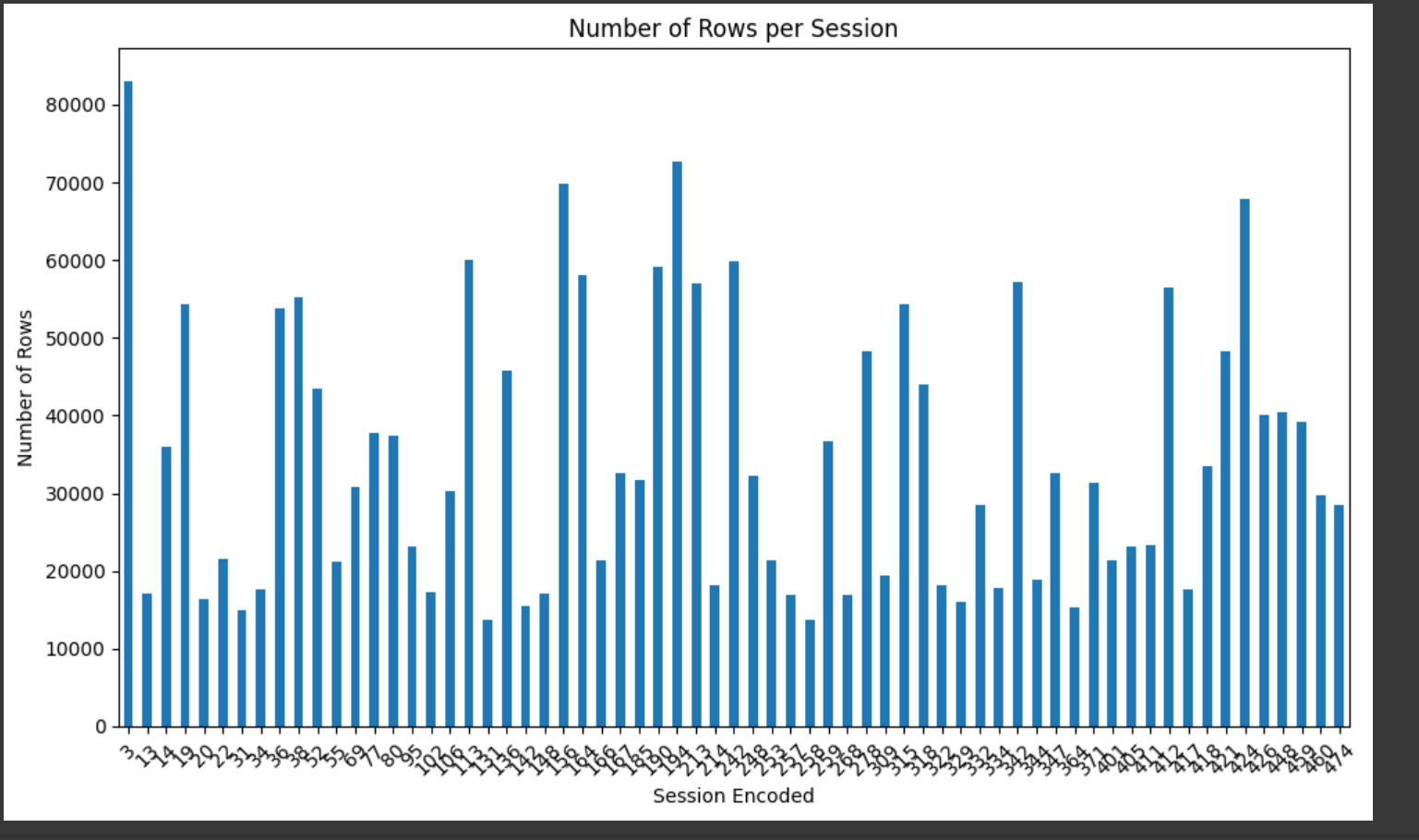


Fig 1.4

We trimmed every session which has timesteps greater than 13456 so that after this step every session will have only 12000 timsteps. After this step, we have obtained the test data from the Balabit mouse data and applied the same preprocessing steps for it. We then went on to reshape our data so that our train data and test data is now in the form B x N x D, where B = Number of independent sessions, N = Length of each session (12000 in this case), D = Dimensions of the data or the columns in the data provided (4 in this case, Because we have also dropped the session\_encoded column as LSTM is taking the input session by session i.e number of sessions is the batch size).

**Shape of the data that is passed to the RNN Model:**

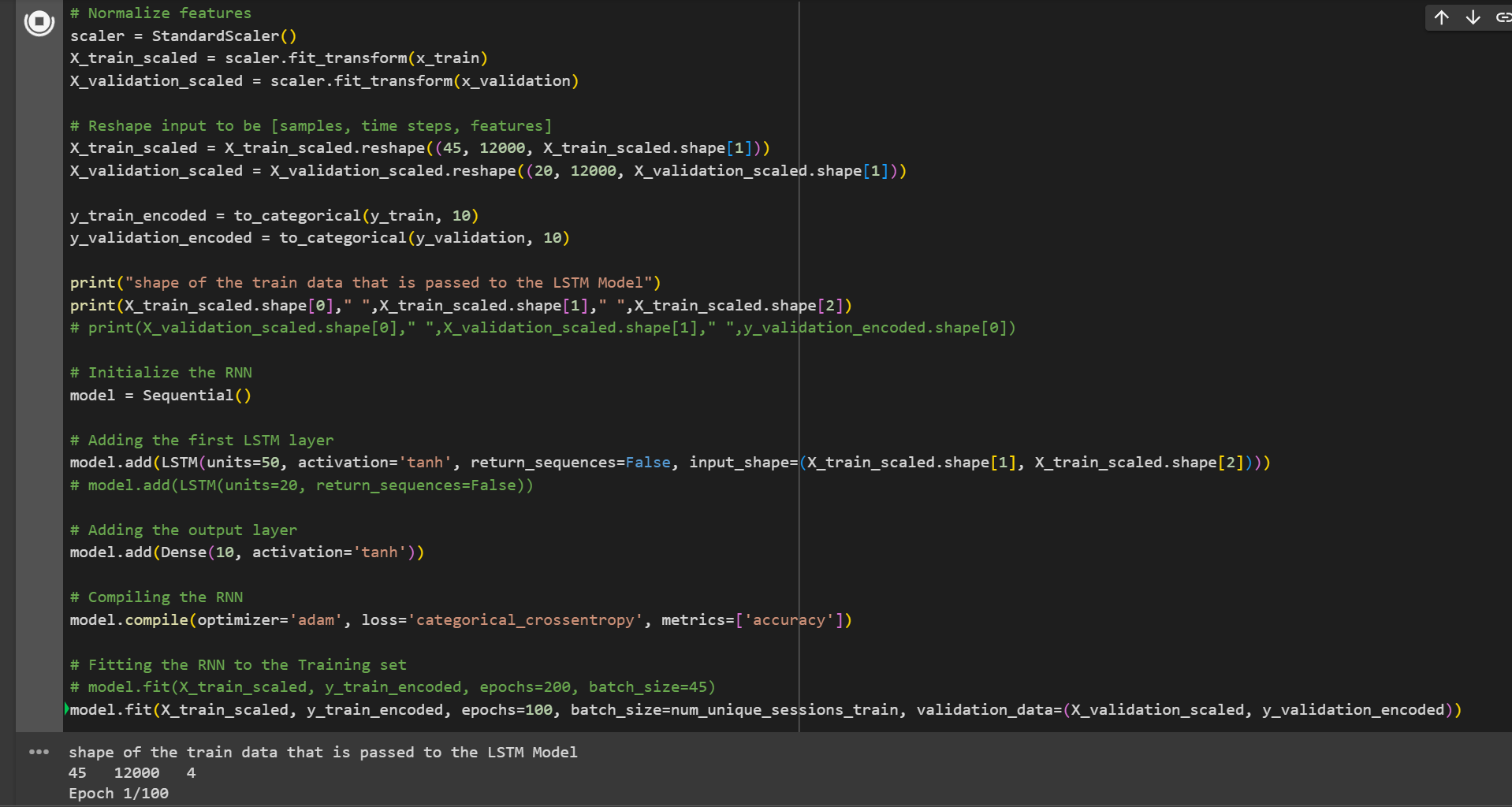


Fig 1.5

### **1.1. Testing the accuracy of the model**

And for testing the accuracy of the model, we have used the *public\_labels.csv* file which looks as shown below.

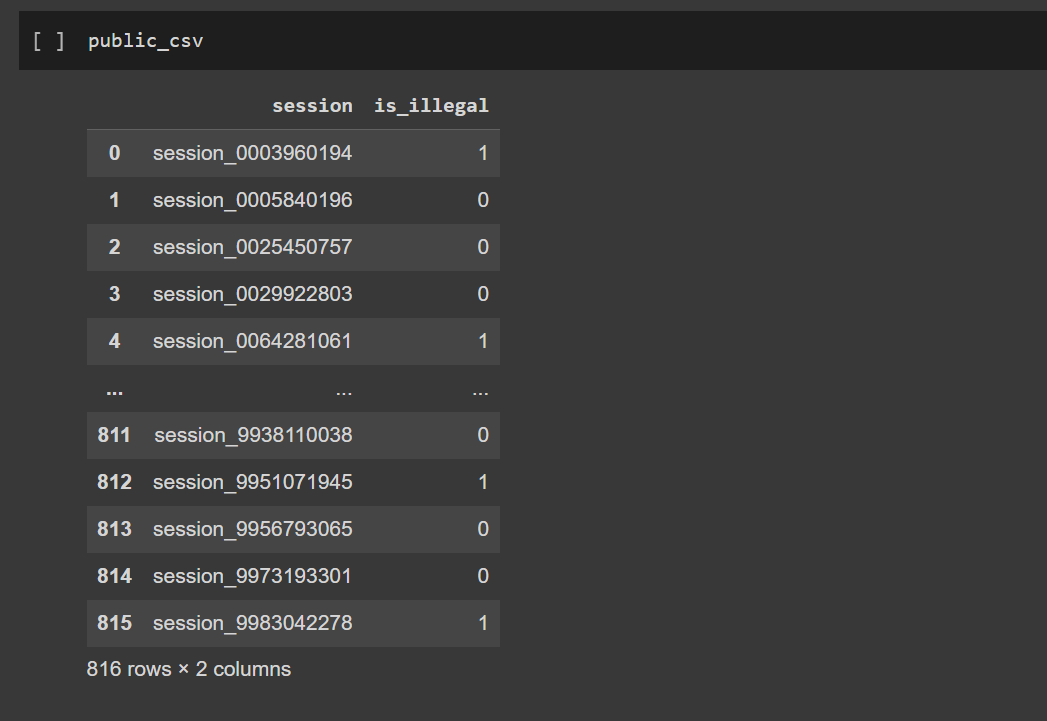


Fig 1.6

We have dropped all the sessions from the test data that are not present in the *public\_labels.csv* file because we are not sure of its validity. Additionally, we also dropped the sessions where the *is\_illegal* column has a value of 1, which means the user is not authorized and unknown. We then predict the user of the session using our model and compare it with the original values (i.e dataframe which has the correct user for each session) and calculate the accuracy of the model.

### **1.2. Split of training and validation data**

|  |  |  |
| --- | --- | --- |
| Validation data count in terms of sessions | Validation Accuracy (%) | Test Accuracy (%) |
| 1 | 80.4 | 70.7 |
| 2 | 80.99 | 70.54 |
| 3 | 82.56 | 75.34 |
| 4 | 81.5 | 67.45 |
| 5 | 79.6 | 69.81 |

From the above table we can observe that the highest test accuracy for this model is obtained when we have taken 3 sessions from each user for validation. Also, when we take 4 or more sessions the validation loss goes up in later epochs which represents **overﬁtting**. **Overfitting** occurs when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means the model will have a low accuracy when predicting outcomes on unseen data, despite having high accuracy on the training data. As the training data set size reduces the model is not able to learn eﬃciently due to high variance and hence the accuracy decreases.

### **1.3. Scaling/No Scaling**

|  |  |  |
| --- | --- | --- |
| Scaling | Validation Accuracy (%) | Test Accuracy (%) |
| No | 30.51 | 29.58 |
| Yes | 80.4 | 71.29 |

It is clear from the results above that feature scaling enhanced the test and validation accuracy. **Feature scaling** is a method to standardize the independent features present in the data in a fixed range. It is done to handle highly varying magnitudes during the pre-processing step. If this step is ignored, then the algorithm tends to weigh greater values, higher and consider smaller values as lower values, regardless of the unit of values. Because **numbers in the input are just numbers, what matters is the relationship amongst them**. Scaling the features makes sure that all features are on a comparable scale and have comparable values.

### **1.4 Activation function in the final layer**

|  |  |  |
| --- | --- | --- |
| Activation Function | Validation Accuracy (%) | Test Accuracy (%) |
| tanh | 22.17 | 22.96 |
| relu | 13.46 | 13.74 |
| softmax | 70.60 | 71.32 |

For the output layer I have chosen softmax as the **softmax activation function is commonly used when you are dealing with a multi-class classification problem. It is particularly useful when you want to assign probabilities to multiple classes,where each class is mutually exclusive** (i.e., an input belongs to exactly one class out of several possible classes). Out of all the combinations we have tried, we have obtained the best accuracy when we used softmax for the output layer and Relu for the rest of the layers.

### **2. BiLSTM**

We have used the same hyperparameters and built the model using BiLSTM. BiLSTM gave better accuracy than LSTM because BiLSTM processes the data in both forward and backward directions. This means it has access to context from both ends of the sequence, leading to a richer understanding of the data. Also, with two sets of weights (forward and backward), BiLSTMs can be more flexible in learning from the sequence data, potentially leading to better performance on complex tasks.

|  |  |  |
| --- | --- | --- |
| BiLSTM | Validation Accuracy (%) | Testing Accuracy (%) |
| Experiment 1 | 79.54 | 73.21 |
| Experiment 2 | 84.21 | 80.52 |
| Experiment 3 | 85.62 | 82.76 |

### **3. GRU**

We have used the same hyperparameters as LSTM and BiLSTM and built the model using GRU. GRU gave lesser accuracy than LSTM because GRUs have two gates (reset and update gates), while LSTMs have three (input, output, and forget gates). This difference can impact how each model processes and retains information over sequences. LSTMs have a separate cell state in addition to the hidden state, providing an additional mechanism to carry information across time steps. This might help LSTMs capture long-term dependencies more effectively in some cases. For very long sequences or sequences with complex structures, LSTMs might perform better due to their additional mechanisms to control the flow of information. Since we have very long sequences in our case, LSTMs performed better than GRU.

|  |  |  |
| --- | --- | --- |
| GRU | Validation Accuracy (%) | Testing Accuracy (%) |
| Experiment 1 | 84.21 | 70.23 |
| Experiment 2 | 83.56 | 68.32 |
| Experiment 3 | 81.85 | 71.76 |

Comparative analysis of models over its accuracy

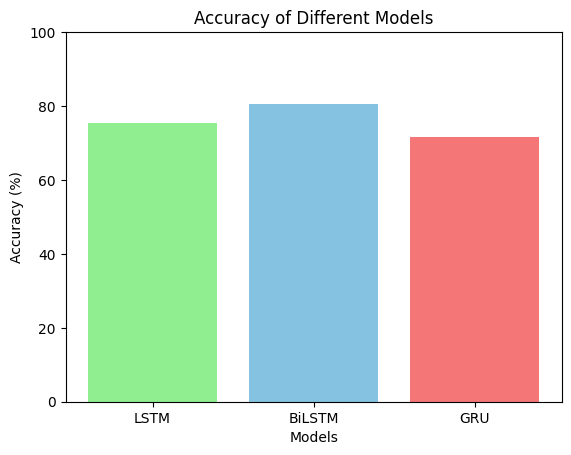


Fig 1.7

### **4. Random Forest algorithm**

Random forest has the adaptability to tackle both regression and classification problems. It has the ability to handle complex datasets and mitigate overfitting. The given mouse dynamics dataset had high-dimensional data. There exists nonlinear interactions between the input features x and y and there exists categorical data for other features like ‘button’, ‘state’ and ‘session’.

The experiment involves preprocessing of dataset where the categorical features undergo encoding with the help of LabelEncoder techniques. The training dataset is divided into training and testing dataset, and the model’s accuracy on the test set is evaluated to establish baseline accuracy. The classifier is trained with 100 trees. There are majorly two hyperparameters involved in n\_estimators which is number of trees and max\_depth which is depth of trees.

Total number of users = 10

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment ID** | **Number of Trees** | **Max Depth** | **Accuracy (%)** |
| Baseline | 100 | - | 78.9 |
| Experiment 1 | 50 | 10 | 54.7 |
| Experiment 2 | 100 | 10 | 53.7 |
| Experiment 3 | 100 | 20 | 59.21 |
| Experiment 4 | 150 | 20 | 40.76 |
| Experiment 5 | 100 | 30 | 61.5 |
| Experiment 6 | 150 | 15 | 46,3 |
| Experiment 7 | 200 | 20 | 37.8 |
| Experiment 8 | 100 | 25 | 55.12 |

In this process, we reshape the target variable that is the user column in a way where we have grouped by session, as done in other experiments.

The baseline achieved an accuracy of 78.9%. Experiment 5, conducted with a parameter setting of 100, 30, yielded the highest accuracy at 61.5%. Conversely, Experiment 7, with parameters 200, 20, demonstrated the lowest accuracy at 37.8%. The results highlight the sensitivity of the model's performance to parameter adjustments, emphasizing the need for careful parameter tuning for optimal outcomes.

### **5. Support vector machine**

SVM is a popular supervised machine learning algorithm used for classification use cases.In our case performs well in high-dimensional spaces, and your dataframe includes several features such as timestamp, client timestamp, x, y, button, state, user, and session. The capability of SVM to be less prone to overfitting and the categorical nature of input feature set , the data can be fed directly without the need of one-hot encoding.

The data is preprocessed and split into training and testing with feature scaling is applied using StandardScaler to ensure that all features have the same scale. We have several hyperparameters that includes kernel, c which is regularization term and and gamma param, altering all three with different values has given with a range of different accuracy for the test data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment ID** | **Kernel** | **C** | **Gamma** | **Accuracy (%)** |
| Baseline | RBF | 1.0 | Auto | 58.5 |
| Experiment 1 | Poly | 0.1 | 0.01 | 57.53 |
| Experiment 2 | RBF | 10.0 | 0.1 | 60.71 |
| Experiment 3 | Sigmoid | 1.0 | Auto | 59.1 |
| Experiment 4 | Poly | 1.0 | 0.1 | 58.4 |
| Experiment 5 | Linear | 0.1 | Auto | 50.12 |
| Experiment 6 | Poly | 1.0 | Scale | 52.7 |
| Experiment 7 | Sigmoid | 0.01 | Auto | 55.98 |
| Experiment 8 | RBF | 5.0 | 0.01 | 60.32 |

The baseline RBF kernel achieved 58.5% accuracy, while Experiment 2, using an RBF kernel with C=10.0 and gamma=0.1, attained the highest accuracy at 60.71%. The results emphasize the impact of kernel choice and parameter tuning on SVM performance, with Experiment 2 showing the most promising outcome.

The RBF and Poly kernels, in particular, exhibit strong performance, emphasizing their suitability while the selection of optimal C and gamma values, remains contingent on the characteristics of the underlying data.

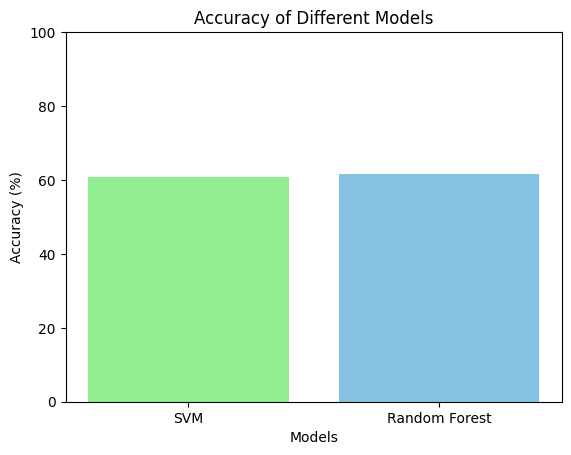
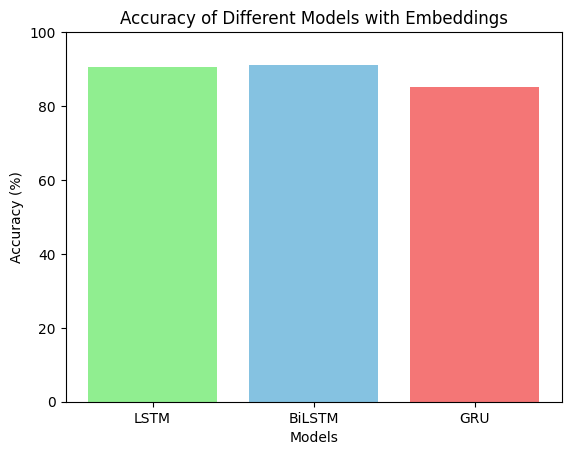


Fig 1.8

**6. BERT**

**We have taken the columns from the dataset (‘record ’‘x’, ‘y’, ‘button’, ‘state’)**

The preprocessed data is tokenized using the RobertaTokenizer. The RobertaTokenizer is initialized using the ‘roberta-large’ model described in section 3.3.3(change later). The batch\_encode\_plus method returns the embeddings in the form of TensorFlow tensors, which are then split into training and testing data. The pre-trained RoBERTa model is loaded using the ‘TFRobertaModel.from\_pretrained’ method and then we combine this with the RNN models GRU, LSTM, and biLSTM. The embeddings are provided to the RNN models, and the RoBERTa model is incorporated into the hybrid model as an additional layer following the input level. The models' layers and parameters are displayed in the table below.

****

## 

## 

## **Conclusion and Future Works**

Among the recurrent neural network (RNN) models, BiLSTM achieved the highest accuracy at 80.67%, followed by LSTM at 75.34%, and GRU at 71.6%. Incorporating embeddings with these RNNs demonstrated notable improvements, with BiLSTM achieving the highest accuracy at 91.2%, followed by LSTM at 90%, and GRU at 85%.

On the other hand, non-RNN models such as SVM and RandomForest showed lower accuracies, with SVM achieving 60.71% accuracy and RandomForest at 61.5%. The results suggest that, for this particular mouse dynamics dataset, the use of BiLSTM with embeddings yielded the most accurate results in identifying unauthorized users, outperforming other models considered in the analysis.

The mouse dynamics dataset is a collection of intricate movement patterns during a human interaction with digital interfaces, so this data can be considered as a distinct behavioral trait. This uniqueness can be captured by temporal data representation using the timestamps, coordinates, click events, velocity, acceleration and so on. The X and Y values correspond to pixel positions in the image. CNN is known best to apply when dealing with spatial data. This deep learning model is well-suited for processing structured grid data like images. The presence of spatial hierarchies and shared weights in CNNs offers the potential to construct a more accurate model.