IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Name: Anuradha Athukorala
Student Reference Number: 10899162

Module Code: PUSL3123	Module Name: AI and Machine Learning
Coursework Title: Group Coursework	
Deadline Date: 15/12/2024	Member of staff responsible for coursework: Dr Neamah Al-Naffakh
Programme: BSc (Hons) Software Engineering	

Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.

Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

Group 85	
A.R.A.N Athukorala	10899162
AATM Saparamadu	10677991
H.B.U. Isuru Senarath	10899692
G.N. Aluthge	10900349
Malwattage Peiris	10899649

We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.

Signed on behalf of the group: Anuradha Athukorala

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***

Signed :

Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.
I *have used/not used translation software.
If used, please state name of software……

Overall mark _____% Assessors Initials _____ Date _____

Table of Contents

1 Introduction

Given the rapid evolution of technology, it is crucial to ensure that only authorized individuals may securely access sensitive information.

The utilization of the internet has evolved due to cellphones, PCs, and several other devices. Individuals increasingly engage in more significant activities online, such as shopping and bill payment. However, these improvements also bring about big security problems, since old ways of authenticating often fail to keep people from getting in without permission. This study goes into detail about how supervised machine learning can be used to create better identification systems that work all the time and are easy to understand.

The main goal of this study is to find out how useful acceleration-based traits could be for using neural networks to verify user identities. Using Feedforward Multi-Layer Perceptrons (feedforwardnet), this study aims to study, train, and improve models that can tell the difference between people based on how they behave in different situations. To do this, the dataset will be carefully looked at to find out the inter- and intra-variance using descriptive statistics, which will put light on the data's natural properties.

This assignment also stresses how important it is to choose the right features and optimize the classifier in order to make the model work better. By using these methods, you can be sure of making a strong and accurate machine learning model that fits the needs of acceleration-based identification. By solving the key challenges associated with user identity verification, this study not only adds to the field of machine learning but also paves the way for the development of new solutions in cybersecurity.

This study explains the methods, results, and effects of employing neural networks for user identification, aiming to show the usefulness and efficiency of machine learning techniques in protecting private digital interactions. Through this exploration, it tries to bridge the gap between technical advancements and security imperatives in the digital age.

2 Literature review.

Biometric-based user authentication uses pattern recognition to recognize or confirm users' behavioral patterns. Authentication is usually referred to as a verification task in mobile security because it confirms the legitimate user based on certain biometric data (Abuhamad et al., 2021). And acceleration-based user authentication is a biometric authentication technique that confirms a user's identity by analyzing their distinct movement patterns using information from motion sensors, especially accelerometers. This method takes advantage of people's propensity for unique gaits which are verified by the analysis of distinct walking patterns. While some methods use wearable cameras to record gait sequences and analyze them using time-series analysis, another uses accelerometer data to extract gait characteristics and detect steady step cycles even at different walking speeds.(Wang et al., 2020).

Furthermore, other than accelerometers and cameras ground reaction force sensors which are wearable devices like GRF sensors measure forces applied during walking are also used to collect these movement patterns. Moreover, systems such as energy harvesting which are systems that use wearable energy-harvesting techniques to analyze gait patterns while reducing battery consumption in wearable devices, speed adaptation which are advanced algorithms that adjust gait recognition systems to perform effectively at varying walking speeds or in uncontrolled environments are used to improve this acceleration-based user authentication systems.(Liu et al., 2022)

One of the recent studies was conducted by applying recurrent neural networks and Long Short-Term Memory (LSTM) to 3-axis accelerations of walking acquired by a smartphone for gait identification and authentication. Where the accelerations during walking of 21 subjects were recorded in two holding situations. One is while placing the smartphone in the pocket and the other is when the subjects are looking at the screen of the smartphone. This study revealed that the performance of this method is better than the conventional method of extracting features from the accelerations but less accurate than biometric authentication methods such as face ID or fingerprint identification.(Watanabe & Kimura, 2020)

Another research was conducted to improve Fingerprint sensors to protect against presentation attacks, especially the puppet attack using Acceleration-based user authentication. This project proposes a system called FINAUTH. This FINAUTH models the fingertip touch characteristics when users apply their fingers to fingerprint sensors. It’s said that this relies upon common built-in sensors to capture instant behavioral characteristics to authenticate different users. This research results were able to confirm by implementing Acceleration-based user authentication to conventional biometric authentication methods, that it can build a more robust authentication method that can verify legitimate users while successfully denying the access requests from unauthorized users with a low false acceptance rate.(*29th USENIX Conference on Security Symposium (SEC’20) : August 12 - 14, 2020*, 2020)

Moreover, another recent study was conducted to offer a comprehensive survey on Behavioral Biometrics and Continuous Authentication technologies for mobile devices. In that research study, it also mentions that gait modality can be efficiently used to transparently and continuously authenticate individuals as this modality sufficiently categorizes different users by their gait characteristics. Furthermore, this study presents an accuracy of 99.95% was achieved by using the Long Short-Term Memory (LSTM) Recurrent Neural Networks classifier while an EER of 0.17% and 0.13% was achieved by applying the Manhattan method and the MLP classifier respectively.(Stylios et al., 2021)

In conclusion, utilizing distinct movement patterns for identification verification, acceleration-based user authentication shows great promise as a reliable biometric method. These systems effectively record and analyze gait data by leveraging motion sensors, including GRF sensors, wearable cameras, and accelerometers. Energy harvesting and speed-adaptive algorithms are examples of advanced techniques that improve system efficiency and are appropriate for dynamic contexts. The promise of gait-based continuous authentication has been highlighted by recent research that use machine learning models, such as LSTM and MLP classifiers, which have demonstrated remarkable accuracy and low error rates.

3 Testing Methodology

The main architecture that is used for the project is a feedforward neural network configured with 2-3 hidden layers and with varying numbers of neurons for each training and testing instance. The training algorithm that was used for this is the Levenberg-Marquardt optimization (trainlm), which we selected for its efficiency in convergence.

The first step in the methodology was the dataset preparation and merging. As there were multiple collections of data sets provided, we concluded that it would be better to merge certain datasets into one and create a mergedData.mat file. This process involved Verifying the existence of each dataset, merging the datasets into a matrix with rows representing data points and columns as features and saving the merged dataset for the use of subsequent stages.

As the datasets that were merged were not normalized and had data redundances, the next step was data normalization and reduction. The normalization was done to scale features to a [0,1] range to improve neural network performance. Furthermore, PCA was employed to handle multicollinearity and reduce dimensions while retaining 95% of the variance. As for the neural network configuration, we defined the FFNN with an input layer with dimensions matching the number of features after PCA. Also, we implemented 2-3 hidden layers with varying number of neurons for each instance and an output layer mapping to the target classes. Moreover, we set a data division for each instance as well with a base setup of training 70%, validation 15% and testing 15% to ensure fair evaluation.

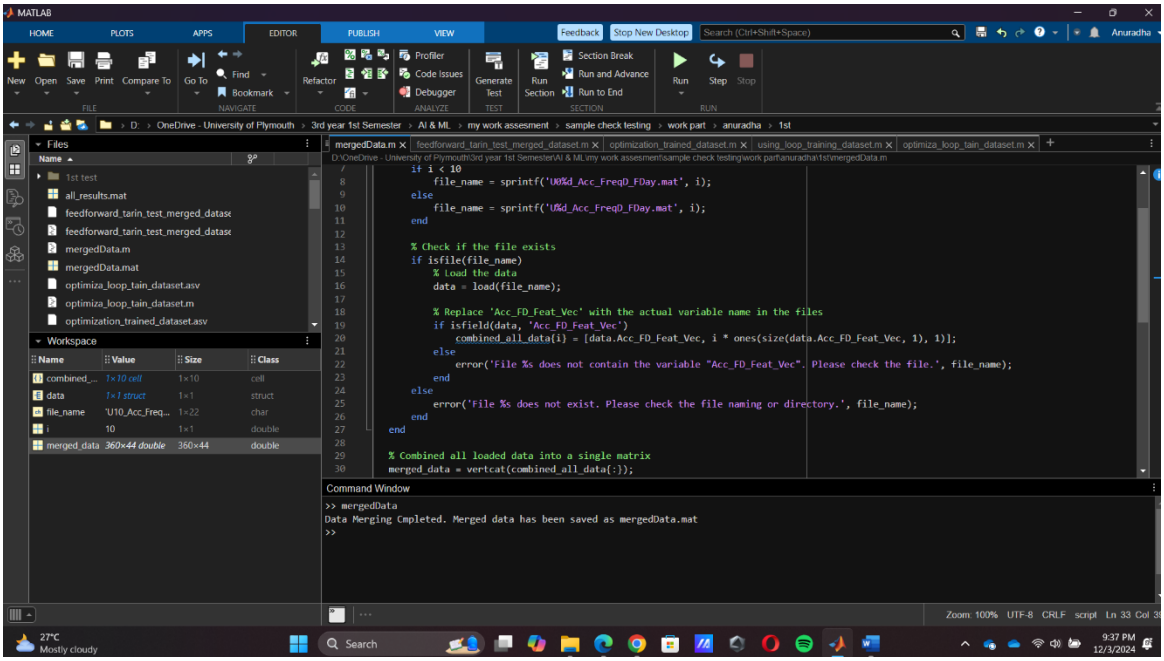
In the training process the FFNN was trained with an initial setup of 1000 epochs as the maximum iteration, a mean squared error (MSE) performance goal of 1e-6 and a minimum gradient threshold of 1e-7 for convergence. Moreover, during training, intra-class and inter-class variances were computed to assess the model's ability to separate data points effectively. And Metrics like training and testing accuracy were figured throughout the testing procedure, which involved comparing the predictions with ground truth labels and evaluating test data against the trained FFNN to ascertain its accuracy.

As mentioned above, for the parameters we took a base set of parameters. The reason for this is that each member of the project took a set of the give data, trained and tested with varying parameters to ensure that to achieve multiple results to see which parameters resulted the best results. Moreover, to ensure reliability, the training and testing process was repeated 10 times using a loop. This was done to create an average accuracy metric and reduced biases due to random initialization.

4 Optimization and Evaluation

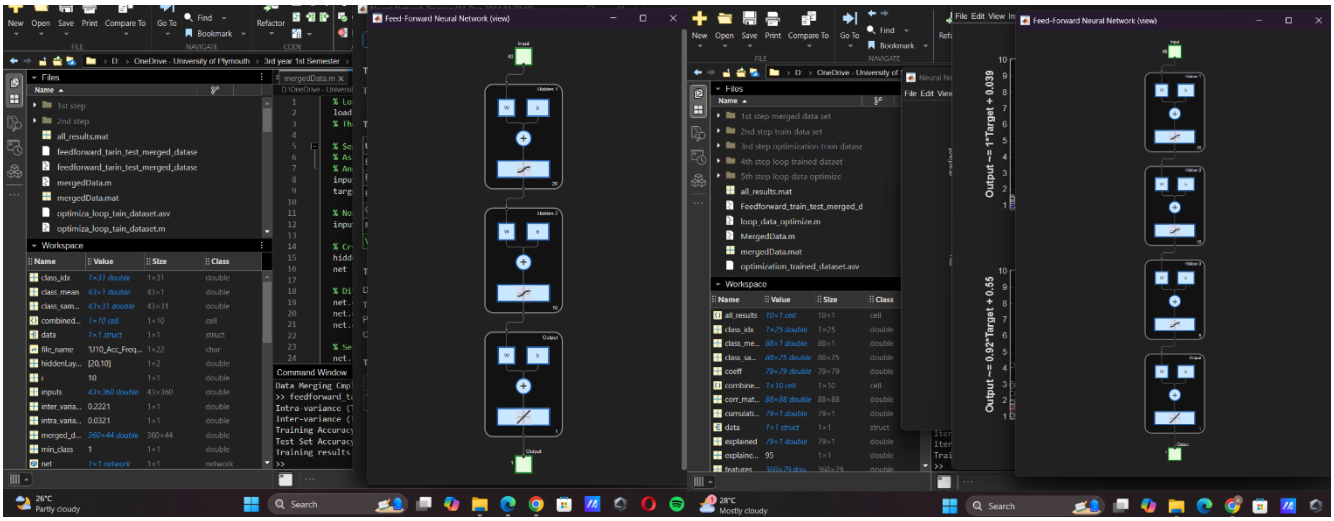
4.1 Merging the datasets

The process began by creating a new folder to organize the dataset files, and all relevant dataset files were added to this folder (these folders were created according to the type which the data was collected. e.g.: - all the U_Acc_TimeD_FreqD_FDay files in one folder). MATLAB was then launched, and the file location was set to the folder containing the datasets. Using MATLAB's Command Window, the data files were loaded into the workspace. To streamline the loading process, a for loop was implemented to load all ten dataset files in a single step efficiently. After the loading process, a verification step was conducted to ensure that all files were loaded correctly. If any files were missing or failed to load, an error message was displayed in the Command Window, assisting prompt identification and resolution of the issue. Once the integrity of the loaded files was confirmed, the datasets were merged into a single comprehensive dataset file. This merged dataset was saved as mergedData.mat. Upon successful completion of the merging process, a "Data merge completed" message was displayed in the Command Window, signifying the successful preparation of the dataset for further processing.

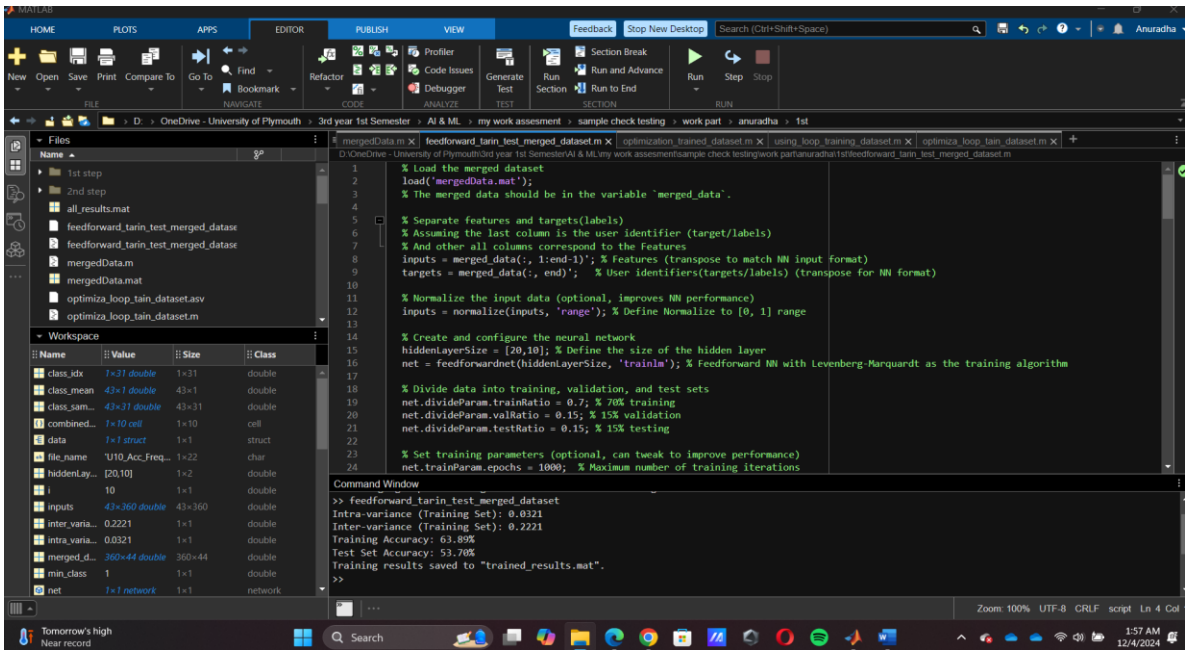


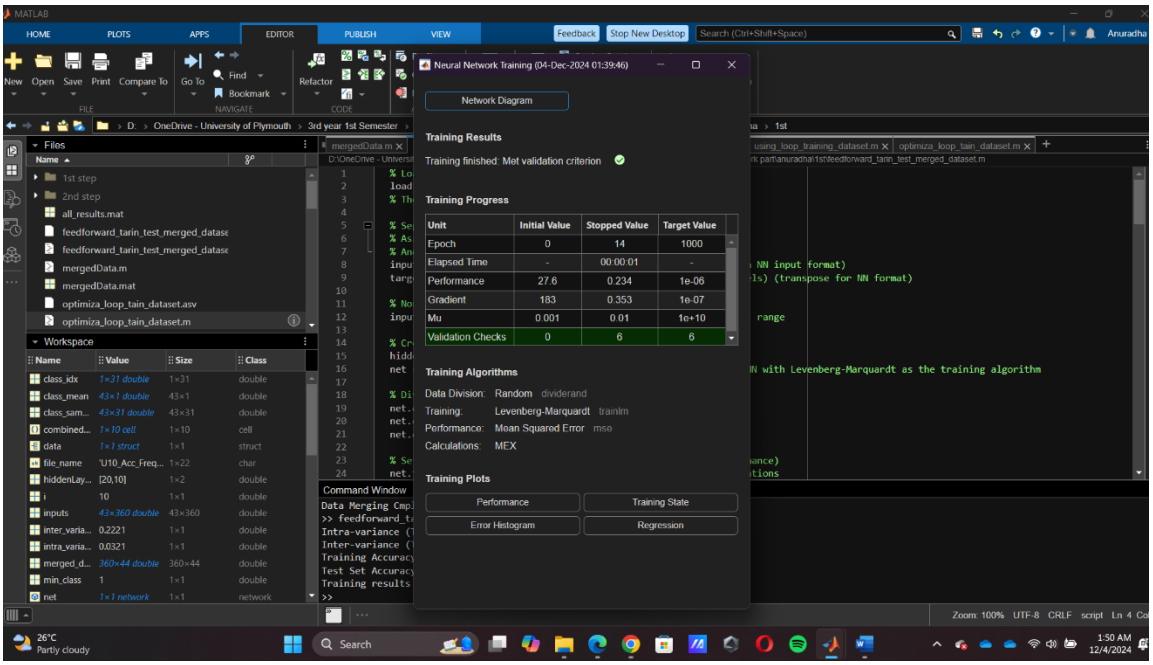
4.2 Feedforward NN, training and testing the datasets.

Once the merged dataset file was created, each merged file was loaded for further processing. The first in this section of the project is the data preparation beginning with separating features and targets, where the last column of the dataset was designated as the target labels, and the remaining columns represented the feature set. To enhance the neural network performance the input features were normalized to the range [0,1]. After that a feedforward NN was then defined and for each training instance we used 1 – 3 hidden layers with varying number of neurons to ensure we can achieve multiple results.



Furthermore, the Levenberg-Marquardt algorithm was selected as the training method due to its efficiency in convergence.





The dataset was divided into training (70%), validation (15%), and test (15%) subsets, ensuring balanced evaluation across these phases. Training parameters were configured to include a maximum of 1000 epochs, a mean squared error (MSE) goal of 1e−6, and a minimum gradient threshold of 1e−7. Moreover, the network was trained using the defined parameters, and intra-variance and inter-variance values were calculated to measure variability within and between classes in the training data. By using loops, variance metrics for each class were calculated and displayed, providing understandings into the distribution of training features.

After training in more than 10 instances for each merged dataset each time with varying parameters, the model's performance was evaluated on both the training and testing datasets. Training accuracy was calculated by comparing predicted and actual classes in the training set, yielding a detailed assessment of the network's ability to learn from the data. Similarly, testing accuracy was determined, ensuring the model's generalizability to unseen data. (some of the results given below)

Furthermore, the intra-variance and inter-variance metrics highlighted the balance of class separability, and the results were saved in a structured format within the trained_results.mat file. Moreover, confusion matrices and performance plots were generated to visually

interpret the model's classification accuracy and error distribution, offering further insights into its effectiveness.

U01_Acc_FreqD_FDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Training Accuracy	63.89%
Test Accuracy	53.70%

U01_Acc_TimeD_MDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Training Accuracy	100.00%
Test Accuracy	77.78%

U01_Acc_TimeD_FDay dataset

	Hidden Layer Size, Validation, Parameters	Training Accuracy	Test Accuracy		Hidden Layer Size, Validation, Parameters	Training Accuracy	Test Accuracy
1st	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	83.33%	6th	60,15 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	68.52%
2nd	20,5 70%, 15%, 15% 1000, 1e-6, 1e-7	95.24%	74.07%	7th	65,10 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	70.37%
3rd	25,10 70%, 15%, 15% 1000, 1e-6, 1e-7	38.10%	25.93%	8th	65,15 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	70.37%
4th	60,10 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	79.63%	9th	65,5 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	83..33%
5th	60	98.41%	61.11%	10th	65	56.35 %	33.33%

	70%, 15%, 15% 1000, 1e-6, 1e-7				70%, 15%, 15% 1000, 1e-6, 1e-7		

U01_Acc_FreqD_MDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Training Accuracy	81.35%
Test Accuracy	75.93%

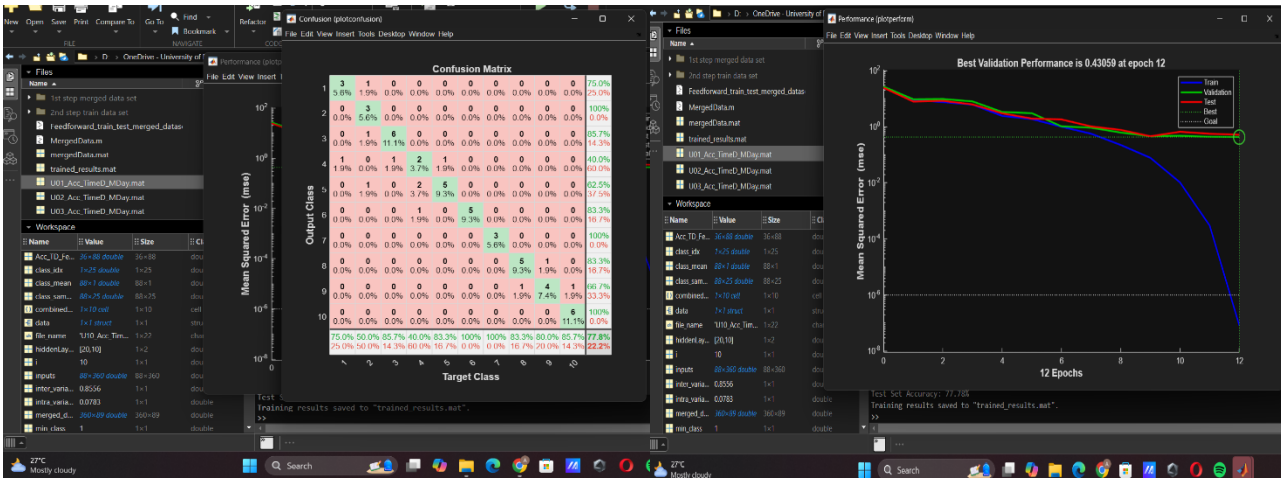
U01_Acc_TimeD_FreqD_FDay dataset

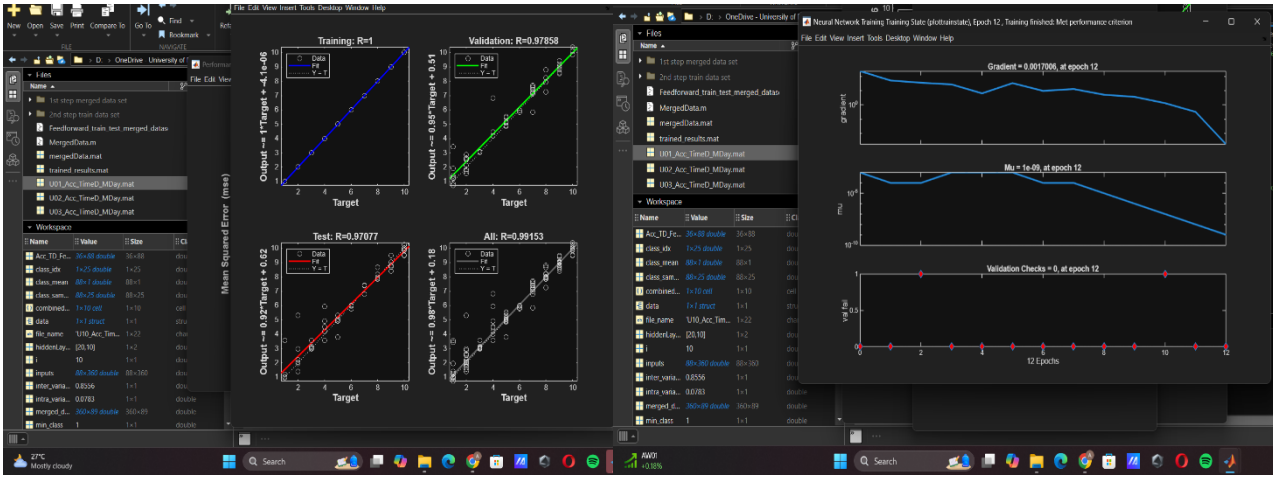
	Hidden layer size, Validation, Parameters	Training accuracy	Test accuracy
1	[20,10] 70%, 15%, 15% 1000, 1e-6, 1e-7	97.62%	75.93%
2	[25,15] 70%, 15%, 15% 1000, 1e-6, 1e-7	100.00%	70.37%
3	[40,10] 70%, 15%, 15% 1000, 1e-6, 1e-7	100.00%	81.48%
4	Hidden layer size, Validation, Parameters	100.00%	66.67%
5	[20,10] 70%, 15%, 15% 1000, 1e-6, 1e-7	86.90%	59.26%

U01_Acc_TimeD_FreqD_MDay dataset

	Hidden Layer Size, Validation, Parameters	Training Accuracy	Test Accuracy		Hidden Layer Size, Validation, Parameters	Training Accuracy	Test Accuracy
1st	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	90.74%	6th	60,15 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	81.48%
2nd	20,5 70%, 15%, 15% 1000, 1e-6, 1e-7	76.19%	61.11%	7th	65,10 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	77.78%
3rd	25,10 70%, 15%, 15% 1000, 1e-6, 1e-7	99.60%	72.22%	8th	65,15 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	79.63%
4th	60,10 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	79.63%	9th	65,5 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	75.93%
5th	60 70%, 15%, 15% 1000, 1e-6, 1e-7	99.21%	68.52%	10th	25,5 70%, 15%, 15% 1000, 1e-6, 1e-7	100%	70.37%

Figures -U01_Acc_TimeD_MDay dataset SS





4.3 Optimization before training dataset

This step is aimed at improving the neural network's performance by addressing feature redundancy and reducing dimensionality through Principal Component Analysis (PCA). After loading the combined dataset and the previously trained model, a correlation matrix was constructed to identify redundant features with strong correlation coefficients ($| r | > 0.99$ |r|>0.99). These redundant characteristics were eliminated to reduce multicollinearity and ensure that the dataset only included unique and significant features. The dataset was then subjected to PCA, with components kept explaining 95% of the variation, yielding a reduced feature set optimal for classification.

Again, as before, the reduced feature set was normalized to the [0,1] range and transposed to correspond to the neural network's input format. A new feedforward neural networks (FFNN) was created using the same hidden layer configurations and the Levenberg-Marquardt algorithm for training. The dataset was separated into three subsets: training (70%), validation (15%), and testing (15%), and the neural network was retrained on the PCA-reduced data.

After the training, the network's performance was checked on both the validation and test datasets. Validation accuracy was calculated to examine the model's generalization during training, whereas test accuracy measured the optimized model's performance on previously unseen data. The modified model attained competitive validation and test accuracies, demonstrating PCA's efficiency in enhancing the classification process.

U01_Acc_FreqD_FDay dataset

Validation Accuracy PCA	79.63%
Test optimization Accuracy PCA	72.22%

U01_Acc_TimeD_MDay dataset

Validation Accuracy PCA	62.96%
Test optimization Accuracy PCA	55.56%

U01_Acc_TimeD_FDay dataset

	Hidden Layer Size, Validation, Parameters	Validation Accuracy After PCA	Test (Optimization) Accuracy After PCA		Hidden Layer Size, Validation, Parameters	Validation Accuracy After PCA	Test (Optimization) Accuracy After PCA
1st	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	70.37%	85.19%	6th	60,15 70%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	61.11%
2nd	20,570%, 15%, 15% 1000, 1e-6, 1e-7	87.04%	85.19%	7th	65,10 70%, 15%, 15% 1000, 1e-6, 1e-7	66.67%	68.52%
3rd	25,1070%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	81.48%	8th	65,15 70%, 15%, 15% 1000, 1e-6, 1e-7	77.78%	66.67%
4th	60,1070%, 15%, 15% 1000, 1e-6, 1e-7	87.04%	79.63%	9th	65,5 70%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	75.93%
5th	6070%, 15%, 15% 1000, 1e-6, 1e-7	48.15%	44.44%	10 th	65 70%, 15%, 15% 1000, 1e-6, 1e-7	66.67%	68.52%

U01_Acc_FreqD_MDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Validation Accuracy PCA	61.11%
Test Accuracy PCA	61.11%

U01_Acc_TimeD_FreqD_FDay dataset

	Hidden layer size, Validation, Parameters	Validation accuracy after PCA	Test (optimization) accuracy after PCA
1	[20,10] 70%, 15%, 15% 1000, 1e-6, 1e-7	3.70%	12.96%
2	[25,15] 70%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	79.63%
3	[40,10] 70%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	74.07%
4	[45,15] 70%, 15%, 15% 1000, 1e-6, 1e-7	66.67%	62.96%
5	[60,10] 70%, 15%, 15% 1000, 1e-6, 1e-7	66.67%	59.70%

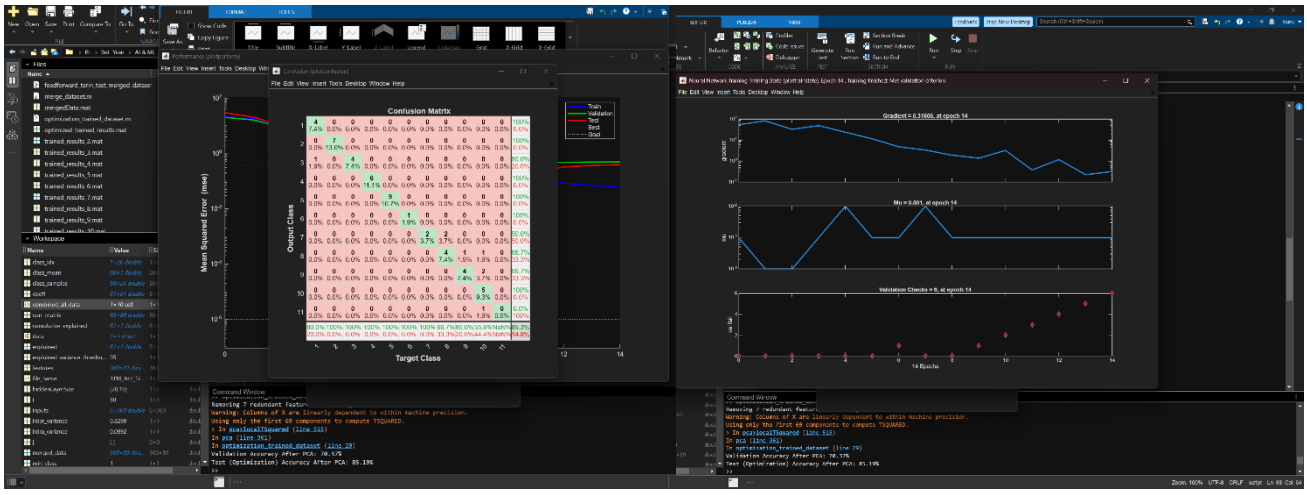
U01_Acc_TimeD_FreqD_MDay dataset

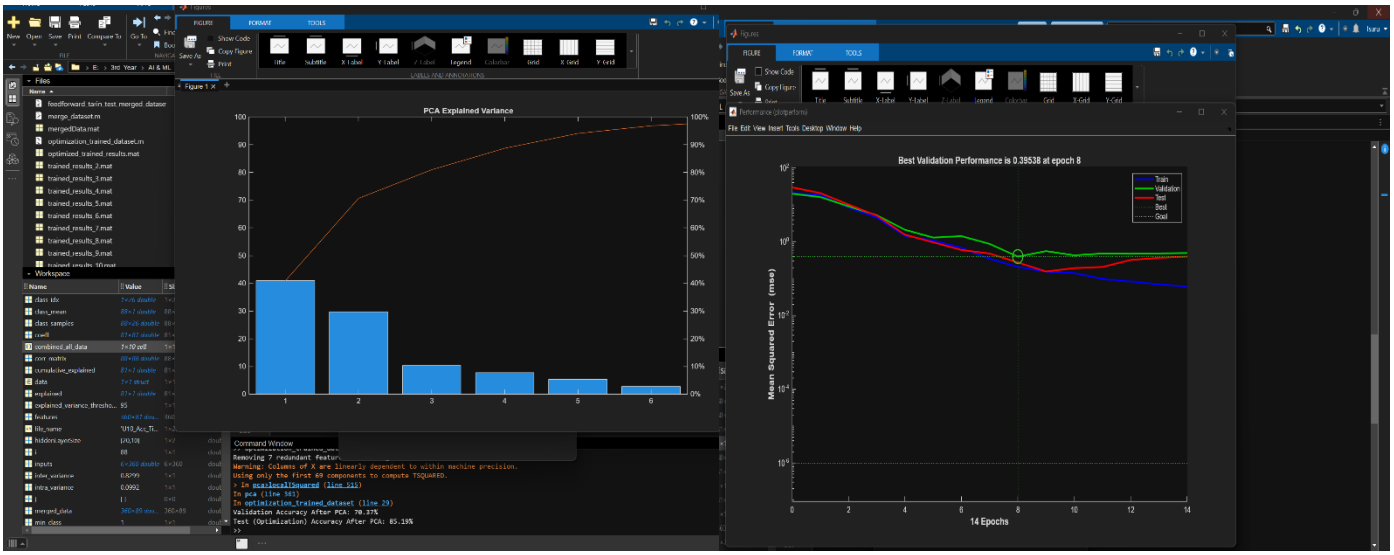
	Hidden Layer Size Validation, Parameters	Validation Accuracy After PCA	Test (Optimization) Accuracy After PCA		Hidden Layer Size	Validation Accuracy After PCA	Test (Optimization) Accuracy After PCA
1st	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	70.37%	68.52%	6th	60,15 70%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	83.33%
2nd	20,5 70%, 15%, 15% 1000, 1e-6, 1e-7	77.78%	61.11%	7th	65,10 70%, 15%, 15% 1000, 1e-6, 1e-7	68.52%	77.78%
3rd	25,10 70%, 15%, 15%	74.07%	72.22%	8th	65,15 70%, 15%, 15%	59.26%	68.52%

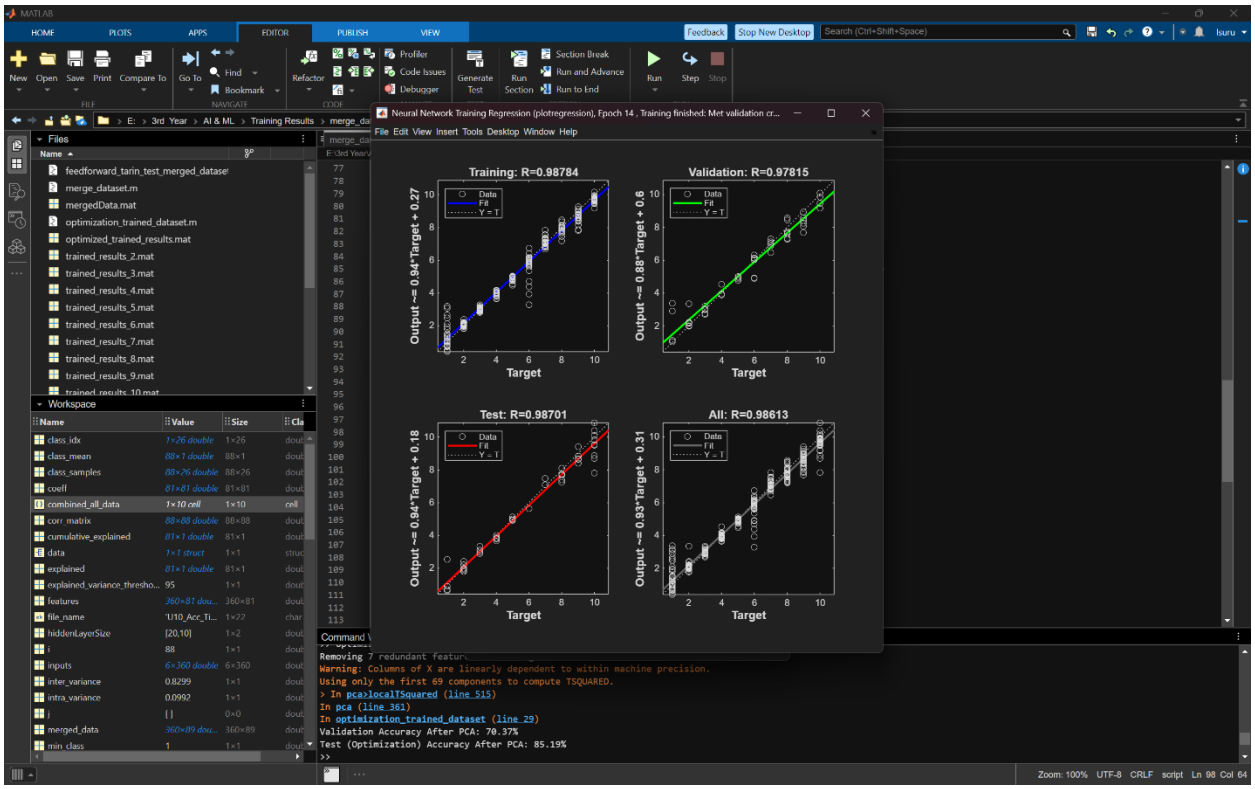
	1000, 1e-6, 1e-7				1000, 1e-6, 1e-7		
4th	60,10 70%, 15%, 15% 1000, 1e-6, 1e-7	72.22%	66.67%	9th	65,5 70%, 15%, 15% 1000, 1e-6, 1e-7	57.41%	57.41%
5th	60 70%, 15%, 15% 1000, 1e-6, 1e-7	74.07%	61.11%	10th	65 70%, 15%, 15% 1000, 1e-6, 1e-7	66.67%	68.52%

The findings, including validation and test accuracy, as well as the number of PCA components employed, were saved in optimized_trained_results.mat for consistency and more analysis. Performance graphs, such as a PCA variance explanation graph and confusion matrices, were created to provide visual insights into the model's optimization. This phase displayed a rigorous approach to feature selection, dimensionality reduction, and model optimization, which resulted in increased accuracy and lower computational complexity.

Figures- U01_Acc_TimeD_FDay dataset




PCA Explained Variance
Best Validation Performance is 0.39538 at epoch 8
Mean Squared Error (mse)
14 Epochs
Train
Validation
Test
Best
Goal
Command Window
Removing 7 redundant feature.
Warning: Columns of X are linearly dependent to within machine precision.
Using only the first 69 components to compute TSQUARED.
In pca (line 361)
In optimization_trained_dataset (line 29)
Validation Accuracy After PCA: 70.37%
Test (Optimization) Accuracy After PCA: 85.19%


Neural Network Training Regression (plotregression), Epoch 14 , Training finished: Met validation cr...
Training: R=0.98784
Validation: R=0.97815
Test: R=0.98701
All: R=0.98613
Output ~= 0.94*Target + 0.27
Output ~= 0.88*Target + 0.6
Output ~= 0.94*Target + 0.18
Output ~= 0.93*Target + 0.31
Target
Data
Fit
Y = T
Removing 7 redundant feature.
Warning: Columns of X are linearly dependent to within machine precision.
Using only the first 69 components to compute TSQUARED.
In pca>localTSquared (line 515)
In pca (line 361)
In optimization_trained_dataset (line 29)
Validation Accuracy After PCA: 70.37%
Test (Optimization) Accuracy After PCA: 85.19%

4.4 Loop training data sets

In this step to ensure robust evaluation and increase the reliability of the training and testing processes, the dataset was trained and tested over ten iterations using a loop-based approach. The major goal was to address potential unpredictability in model performance owing to random initializations and data splits.

To improve the neural network's performance, the mergedData.mat file was loaded for each iteration, and the features and targets were normalized. The Levenberg-Marquardt algorithm was again used to train a feedforward neural network by using one of the hidden layer parameters that gave the best results in the previous steps. For every iteration, the dataset was split into subsets for training (70%) and validation (15%) and testing (15%). To guarantee efficient learning, training parameters were established, including a maximum of 1000 epochs and a performance target of 10^ −6.

Each iteration's training and test accuracies were calculated. While test accuracy evaluated the model's performance on unseen data, training accuracy analyzed the model's performance on the training subset. For every iteration, these accuracies were saved in separate result files (trained_results_i.mat) and shown in the command window. Furthermore, for additional analysis, all training and testing accuracy data were recorded in a single file (all_results.mat).

U01_Acc_FreqD_FDay dataset

	1st time	2nd time	3rd time	4th time	5th time
Hiddenlayers, Validation, Parameters	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 70%, 15%, 15% 1000, 1e-6, 1e-7	20,10 70%, 15%, 15% 500, 1e-5, 1e-8	20,10 60%, 20%, 20% 1000, 1e-6, 1e-7	30,15 60%, 20%, 20% 500, 1e-5, 1e-8
Training Accuracy	98.81%	82.54%	89.29%	89.81%	78.24%
Testing Accuracy	81.48%	83.33%	85.19%	73.61%	79.17%

U01_Acc_TimeD_MDay dataset

	1st time	2nd time	3rd time	4th time	5th time
Hiddenlayers, Validation, Parameters	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 60%, 20%, 20% 500, 1e-5, 1e-8	40,20 80%, 30%, 30% 1000, 1e-6, 1e-7	20,10, 15 70%, 30%, 30% 1000, 1e-7, 1e-9	40,20,10 70%, 20%, 20% 1000, 1e-6, 1e-7
Training Accuracy	94.05%	100%	100%	97.94%	83.48%

Testing Accuracy	85.19%	77.78%	67.53%	84.34%	67.69%

U01_Acc_TimeD_FDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Training Accuracy	78.33%
Testing Accuracy	75.37%

U01_Acc_FreqD_MDay dataset

	1st time	2nd time	3rd time	4th time	5th time
HiddenLayers, Validation, Parameters	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 70%, 15%, 15% 1000, 1e-6, 1e-7	20,10 70%, 15%, 15% 500, 1e-7, 1e-8	20,10 60%, 20%, 20% 500, 1e-7, 1e-8	30,15 50%, 10%, 10% 500, 1e-6, 1e-7
Training Accuracy	67.07%	64.90%	80.91%	71.60%	72.64%
Testing Accuracy	64.25%	57.91%	69.26%	48.91%	50.03%

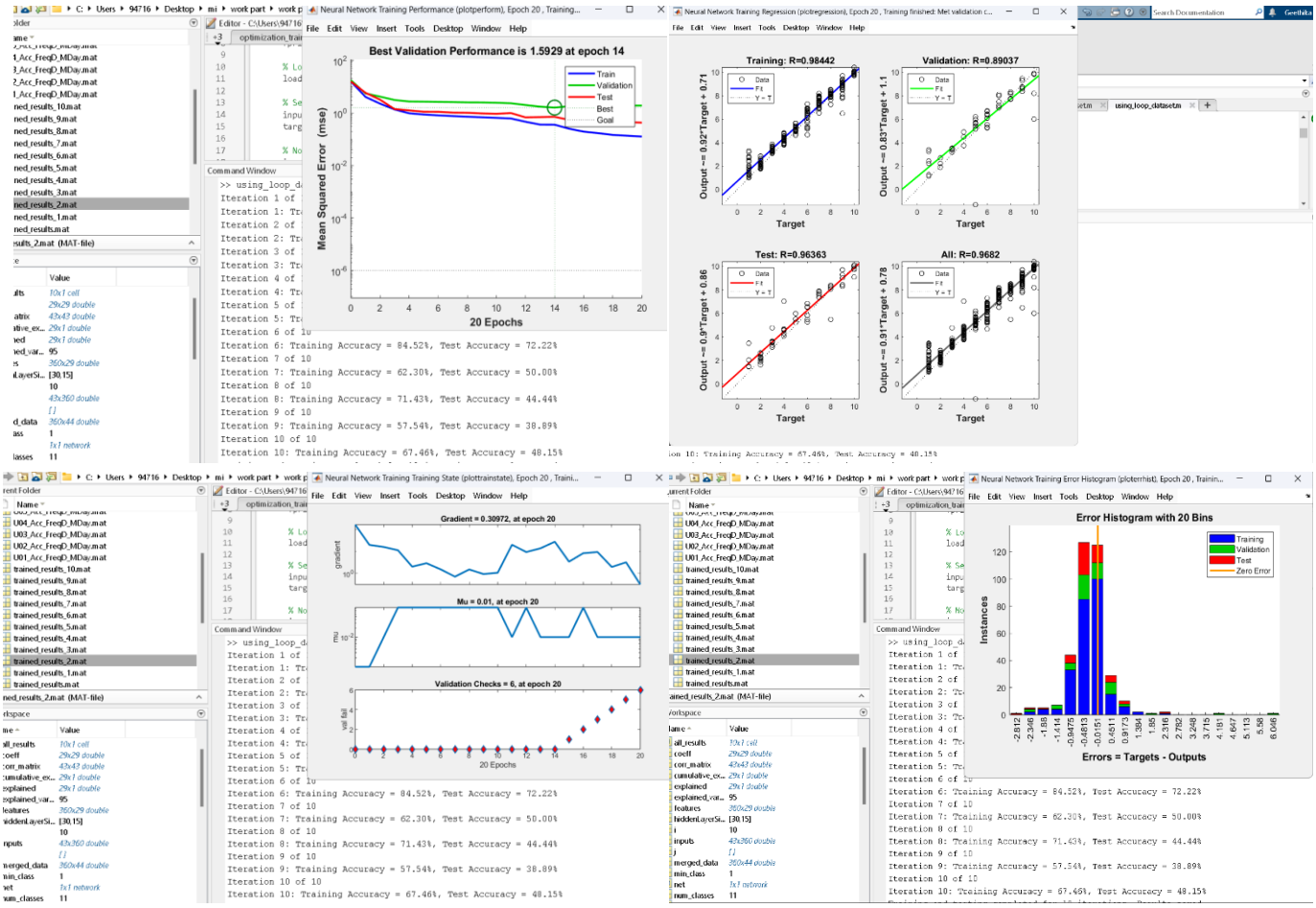
U01_Acc_TimeD_FreqD_FDay dataset

Hidden layers	40,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Training Accuracy	88.89%
Testing Accuracy	72.22%

U01_Acc_TimeD_FreqD_MDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Training Accuracy	97.62%
Testing Accuracy	83.33%

Figures- U01_Acc_FreqD_MDay dataset



Due to this step a more complete assessment of the model's generalization skills was provided here by this iterative process, which made sure that performance measurements considered differences in data splitting and network initialization. The outcomes showed steady trends in accuracy throughout iterations, proving the trained neural network's durability and dependability.

4.5 Optimization of the loop trained dataset

Here, the dataset was optimized using Principal Component Analysis (PCA) to further improve the model's performance and decrease dimensionality. First, using a correlation criterion (absolute correlation > 0.99), duplicated characteristics were found and removed. By lowering multicollinearity, this step reduced the feature set. In order to preserve 95% of the explained variation and identify important patterns in the data, PCA was applied to the remaining features. Before being transposed for neural network compatibility, the reduced feature set was normalized to guarantee consistent scaling.

Again, the neural network with hidden layer that gave the best results in previous steps were used in the configuration and was trained and tested across 10 iterations. In each iteration, the dataset was split into training (70%), validation (15%), and testing (15%) subsets. The network was also trained using the Levenberg-Marquardt algorithm with a maximum of 1000 epochs and a performance goal of 10^−6. Validation and test accuracy were calculated in each iteration and stored for analysis.

U01_Acc_FreqD_FDay dataset

Hiddenlayers, Validation, Parameters	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 70%, 15%, 15% 1000, 1e-6, 1e-7	20,10 70%, 15%, 15% 500, 1e-5, 1e-8	20,10 60%, 20%, 20% 1000, 1e-6, 1e-7	30,15 60%, 20%, 20% 500, 1e-5, 1e-8
Validation Accuracy	76.85%	71.11%	76.85%	71.11%	71.53%
Testing Accuracy	71.48%	68.15%	71.48%	66.94%	65.97%

U01_Acc_TimeD_MDay dataset

	1st time	2nd time	3rd time	4th time	5th time
Hiddenlayers, Validation, Parameters	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 60%, 20%, 20% 500, 1e-5, 1e-8	40,20 80%, 30%, 30% 1000, 1e-6, 1e-7	20,10, 15 70%, 30%, 30% 1000, 1e-7, 1e-9	40,20,10 70%, 20%, 20% 1000, 1e-6, 1e-7
Validation Accuracy	68.89%	70.28%	68.96%	65.54%	75.08%
Testing Accuracy	73.28%	69.72%	66.62%	68.80%	72.62%

U01_Acc_TimeD_FDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Validation Accuracy	93.17%
Testing Accuracy	74.26%

U01_Acc_FreqD_MDay dataset

HiddenLayers, Validation, Parameters	20,10 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 70%, 15%, 15% 1000, 1e-6, 1e-7	30,15 70%, 15%, 15% 700, 1e-7, 1e-8	30,15 80%, 10%, 10% 700, 1e-7, 1e-8	40,20 60%, 20%, 20% 800, 1e-5, 1e-6
Validation Accuracy	62.41%	57.41%	62.96%	68.89%	60.56%
Testing Accuracy	59.81%	61.30%	66.30%	67.22%	65.00%

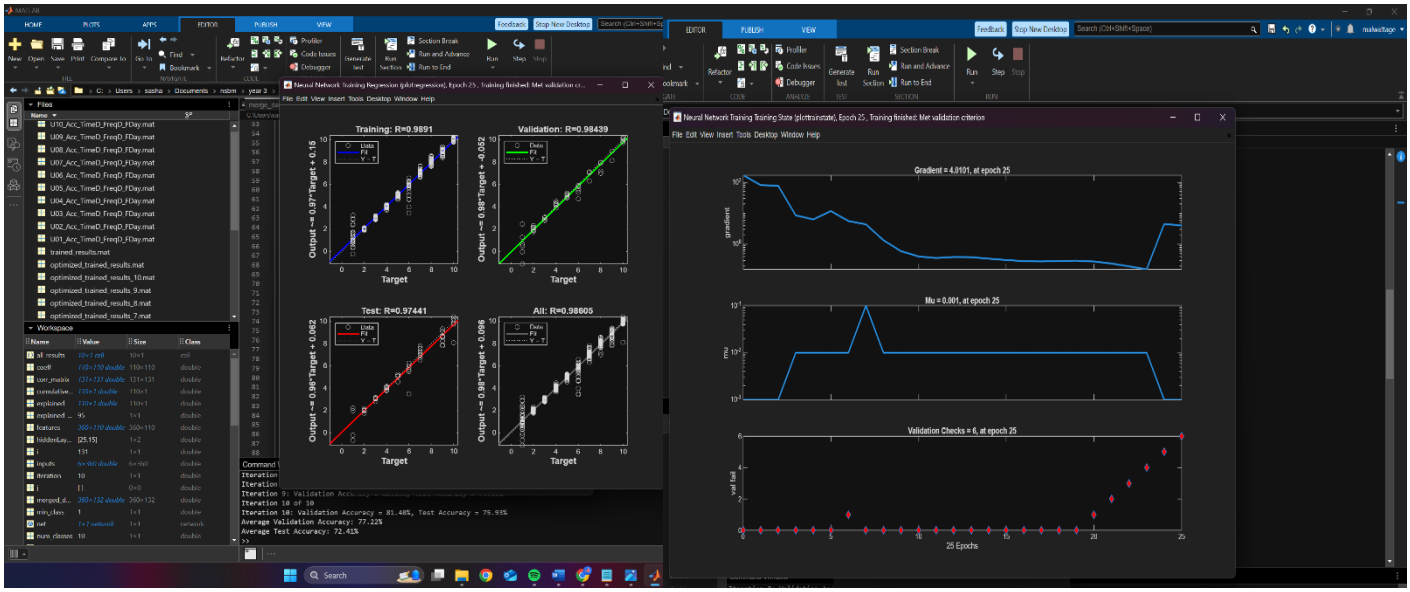
U01_Acc_TimeD_FreqD_FDay dataset

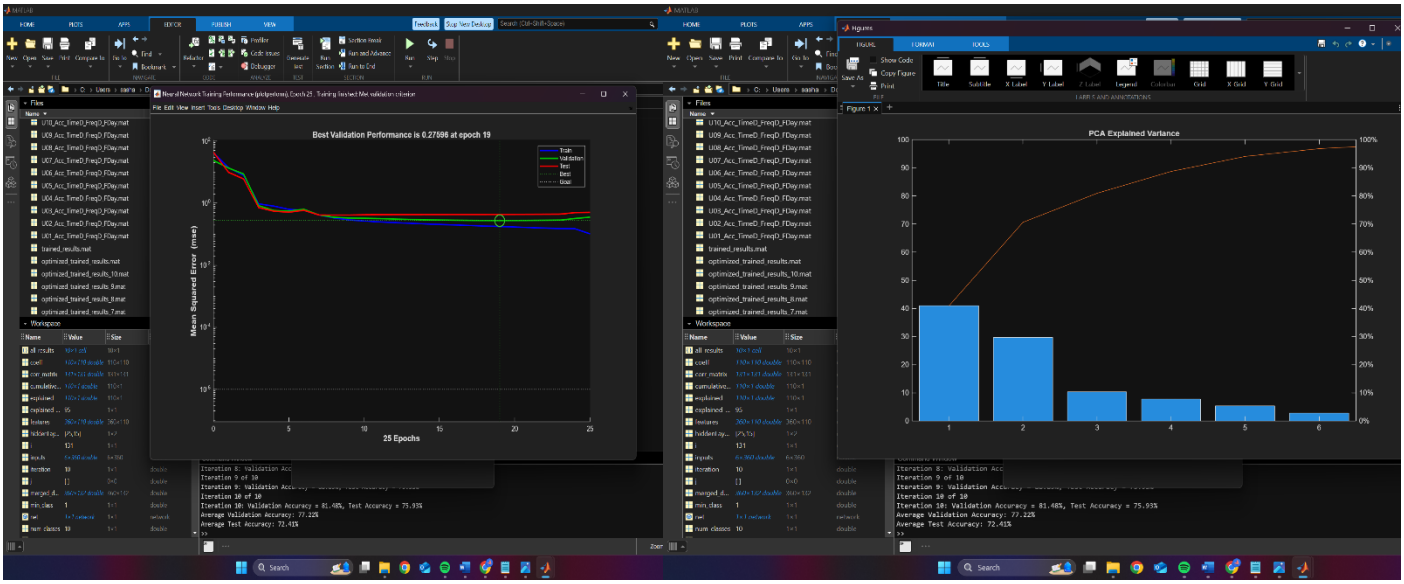
Hidden layers	25,15
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Validation Accuracy	77.22%
Testing_Accuracy	72.41%

U01_Acc_TimeD_FreqD_MDay dataset

Hidden layers	20,10
Validation	70%, 15%, 15%
Parameters	1000, 1e-6, 1e-7
Validation Accuracy	71.48%
Testing_Accuracy	73.89%

Figures- U01_Acc_TimeD_FreqD_FDay dataset




Best Validation Performance is 0.27596 at epoch 19
Mean Squared Error (mse)
25 Epochs
Train
Validation
Test
Best
Goal
PCA Explained Variance
Iteration 10 of 10
Iteration 10: Validation Accuracy = 81.48%, Test Accuracy = 75.93%
Average Validation Accuracy: 77.22%
Average Test Accuracy: 72.41%

5 Conclusion

Overall, the project revealed an effective approach to processing and optimizing datasets for neural network-based classification tasks. Initially, datasets were merged, normalized, and trained on a feedforward neural network with different hidden layers and the Levenberg-Marquardt method. Dimensionality reduction tackles such as PCA enhanced computing efficiency and model accuracy by removing duplication while maintaining important characteristics. Iterative training proved the model's robustness, resulting in consistent performance across trials. Optimization improved accuracy while reducing complexity, demonstrating the model's generalizability. Overall, this structured approach successfully used machine learning techniques to provide a dependable and fast solution for data classification problems.

6 References

29th USENIX Conference on Security Symposium (SEC'20) : August 12 - 14, 2020. (2020). USENIX Association.

Abuhamad, M., Abusnaina, A., Nyang, D., & Mohaisen, D. (2021). Sensor-Based Continuous Authentication of Smartphones' Users Using Behavioral Biometrics: A Contemporary Survey. *IEEE Internet of Things Journal*, *8*(1), 65–84. https://doi.org/10.1109/JIOT.2020.3020076

Liu, S., Shao, W., Li, T., Xu, W., & Song, L. (2022). Recent advances in biometrics-based user authentication for wearable devices: A contemporary survey. *Digital Signal Processing*, *125*, 103120. https://doi.org/10.1016/J.DSP.2021.103120

Stylios, I., Kokolakis, S., Thanou, O., & Chatzis, S. (2021). Behavioral biometrics & continuous user authentication on mobile devices: A survey. *Information Fusion*, *66*, 76–99. https://doi.org/10.1016/J.INFFUS.2020.08.021

Wang, C., Wang, Y., Chen, Y., Liu, H., & Liu, J. (2020). User authentication on mobile devices: Approaches, threats and trends. *Computer Networks*, *170*, 107118. https://doi.org/10.1016/J.COMNET.2020.107118

Watanabe, Y., & Kimura, M. (2020). Gait identification and authentication using LSTM based on 3-axis accelerations of smartphone. *Procedia Computer Science*, *176*, 3873–3880. https://doi.org/10.1016/J.PROCS.2020.09.001

7 Appendix

7.1 1st code file Merged dataset

```
% create an empty cell array to store the data from all file
combined_all_data = {};

% Loop through the files and combine their data
for i = 1:10
    % Create the file name with leading zeros for numbers 1 to 9
    if i < 10
        file_name = sprintf('U0%d_Acc_FreqD_FDay.mat', i);
    else
        file_name = sprintf('U%d_Acc_FreqD_FDay.mat', i);
    end

    % Check if the file exists
    if isfile(file_name)
        % Load the data
        data = load(file_name);

        % Replace 'Acc_FD_Feat_Vec' with the actual variable name in the files
        if isfield(data, 'Acc_FD_Feat_Vec')
            combined_all_data{i} = [data.Acc_FD_Feat_Vec, i * ones(size(data.Acc_FD_Feat_Vec,
1), 1)];
        else
            error('File %s does not contain the variable "Acc_FD_Feat_Vec". Please check the file.',
file_name);
        end
```

```
    else
        error('File %s does not exist. Please check the file naming or directory.', file_name);
    end
end

% Combined all loaded data into a single matrix
merged_data = vertcat(combined_all_data{:});

% Save the merged data to a new file
save('mergedData.mat', 'merged_data');

% Display message
disp('Data Merging Cmpleted. Merged data has been saved as mergedData.mat');
```

7.2 2nd code file Create feedforward NN and train, test, data set

```
% Load the merged dataset
load('mergedData.mat');
% The merged data should be in the variable `merged_data`.

% Separate features and targets(labels)
% Assuming the last column is the user identifier (target/labels)
% And other all columns correspond to the Features
inputs = merged_data(:, 1:end-1)'; % Features (transpose to match NN input format)
targets = merged_data(:, end)';   % User identifiers(targets/labels) (transpose for NN format)

% Normalize the input data (optional, improves NN performance)
inputs = normalize(inputs, 'range'); % Define Normalize to [0, 1] range
```

```
% Create and configure the neural network
hiddenLayerSize = [20,10]; % Define the size of the hidden layer
net = feedforwardnet(hiddenLayerSize, 'trainlm'); % Feedforward NN with Levenberg-Marquardt as the training algorithm

% Divide data into training, validation, and test sets
net.divideParam.trainRatio = 0.7; % 70% training
net.divideParam.valRatio = 0.15; % 15% validation
net.divideParam.testRatio = 0.15; % 15% testing

% Set training parameters (optional, can tweak to improve performance)
net.trainParam.epochs = 1000;  % Maximum number of training iterations
net.trainParam.goal = 1e-6;    % Performance goal (MSE)
net.trainParam.min_grad = 1e-7; % Minimum gradient

% Train the neural network
[net, tr] = train(net, inputs, targets);

% Evaluate intra-variance and inter-variance for the training set
% Extract training set inputs and targets
train_inputs = inputs(:, tr.trainInd);  % Features of training set
train_targets = targets(tr.trainInd);  % Targets of training set

% Get unique classes in the training set
unique_classes = unique(train_targets);
```

```
% Initialize intra-variance and inter-variance
intra_variance = 0;
inter_variance = 0;

% Calculate overall mean of the training features
overall_mean = mean(train_inputs, 2); % Overall mean across all features (column-wise)

% Total number of samples in the training set
total_samples = size(train_inputs, 2);

% use loop calculate variance metrics for each class
for i = 1:length(unique_classes)
    % Find the indices of samples belonging to the current class
    class_idx = find(train_targets == unique_classes(i));

    % Extract features of the current class
    class_samples = train_inputs(:, class_idx);

    % Number of samples in this class
    num_samples_in_class = size(class_samples, 2);

    % Calculate the mean of the current class
    class_mean = mean(class_samples, 2); % Mean across features for this class

    % Compute intra-variance (variance within the class)
    intra_variance = intra_variance + ...
        sum(sum((class_samples - class_mean).^2)) / total_samples;
```

```
    % Compute inter-variance (variance of class means relative to the overall mean)
    inter_variance = inter_variance + ...
        num_samples_in_class * sum((class_mean - overall_mean).^2) / total_samples;
end

% Display inter and intra variance results
fprintf('Intra-variance (Training Set): %.4f\n', intra_variance);
fprintf('Inter-variance (Training Set): %.4f\n', inter_variance);

% Evaluate training accuracy for the training set
train_outputs = net(inputs(:, tr.trainInd)); % Network predictions for training set
train_predicted_classes = round(train_outputs); % Round predictions to nearest integer
train_actual_classes = targets(tr.trainInd);    % Actual training targets

% Calculate training accuracy
train_correct_predictions = sum(train_predicted_classes == train_actual_classes);
train_total_samples = length(train_actual_classes);
train_accuracy = (train_correct_predictions / train_total_samples) * 100;
fprintf('Training Accuracy: %.2f%%\n', train_accuracy);

% Evaluate test accuracy
test_inputs = inputs(:, tr.testInd); % Inputs for the test set
test_targets = targets(tr.testInd); % Targets for the test set

% Generate predictions for the test set
test_outputs = net(test_inputs);
```

```
test_predicted_classes = round(test_outputs);
test_actual_classes = test_targets;

% Ensure classes are positive integers starting from 1
% Shift classes to be 1-based if they start at 0 or contain negative values
min_class = min([test_actual_classes, test_predicted_classes]);
if min_class <= 0
    test_actual_classes = test_actual_classes - min_class + 1;
    test_predicted_classes = test_predicted_classes - min_class + 1;
end

% Calculate test accuracy
test_correct_predictions = sum(test_predicted_classes == test_actual_classes);
test_total_samples = length(test_actual_classes);
test_accuracy = (test_correct_predictions / test_total_samples) * 100;
fprintf('Test Set Accuracy: %.2f%%\n', test_accuracy);

% Plot performance and confusion matrix
figure;
plotperform(tr); % Performance plot (training, validation, test errors)

% One-hot encode the test targets and predicted classes for confusion matrix
num_classes = max([test_actual_classes, test_predicted_classes]); % Determine the number of unique classes
test_actual_classes_onehot = ind2vec(test_actual_classes, num_classes); % One-hot encode actual classes
test_predicted_classes_onehot = ind2vec(test_predicted_classes, num_classes); % One-hot encode predicted classes
```

```
% Plot confusion matrix using one-hot encoded labels
figure;
plotconfusion(test_actual_classes_onehot, test_predicted_classes_onehot); % Confusion matrix

% Save the trained results and model
results.train_accuracy = train_accuracy;   % Training accuracy
results.test_accuracy = test_accuracy;     % Test accuracy
results.intra_variance = intra_variance;   % Intra-variance
results.inter_variance = inter_variance;   % Inter-variance
results.net = net;                         % Trained neural network
results.training_record = tr;              % Training record (training, validation, test performance)

save('trained_results.mat', 'results');    % Save results
fprintf('Training results saved to "trained_results.mat".\n');
```

7.3 3rd code file Optimization before trained data set

```
% Load the trained dataset and  per-tarined model
load('mergedData.mat'); % Load the dataset
load('trained_results.mat'); % Load pre-trained model

% Separate features and targets from the loaded data
features = merged_data(:, 1:end-1); % Features
targets = merged_data(:, end);     % Targets/labels (user identifiers)

% Handle multicollinearity (optional)
```

```
corr_matrix = corrcoef(features); % Correlation matrix
% Find indices of redundant features
redundant_columns = [];
for i = 1:size(corr_matrix, 1)
    for j = i+1:size(corr_matrix, 2)
        if abs(corr_matrix(i, j)) > 0.99
            redundant_columns = [redundant_columns, j]; % Collect the column indices
        end
    end
end
redundant_columns = unique(redundant_columns); % Ensure unique column indices

% Remove redundant features
if ~isempty(redundant_columns)
    fprintf('Removing %d redundant features due to high correlation.\n',
length(redundant_columns));
    features(:, redundant_columns) = [];
end

% Apply PCA to the dataset
[coeff, pca_features, ~, ~, explained] = pca(features); % Perform PCA

% Retain enough components to explain 95% variance
explained_variance_threshold = 95;
cumulative_explained = cumsum(explained);
num_components = find(cumulative_explained >= explained_variance_threshold, 1);
```

```
% Select reduced features
optimized_features = pca_features(:, 1:num_components);

% Normalize the reduced features
optimized_features = normalize(optimized_features, 'range');

% Transpose data for the neural network input format
inputs = optimized_features'; % Transpose features
targets = targets';           % Transpose targets

% Retrain the Neural Network with Reduced Features
hiddenLayerSize = [20, 10]; % Hidden layer configuration
net = feedforwardnet(hiddenLayerSize, 'trainlm'); % Create a new NN

% Divide data into training, validation, and test sets
net.divideParam.trainRatio = 0.7; % 70% training
net.divideParam.valRatio = 0.15;  % 15% validation
net.divideParam.testRatio = 0.15; % 15% testing

% Set training parameters
net.trainParam.epochs = 1000;  % Maximum training iterations
net.trainParam.goal = 1e-6;    % Performance goal (MSE)
net.trainParam.min_grad = 1e-7; % Minimum gradient

% Train the network with PCA-reduced inputs
[net, tr] = train(net, inputs, targets);
```

```
% Evaluate Validation Accuracy
val_inputs = inputs(:, tr.valInd); % Validation inputs
val_targets = targets(tr.valInd);  % Validation targets

val_outputs = net(val_inputs); % Get predictions
val_predicted_classes = round(val_outputs); % Round to nearest integer
val_actual_classes = val_targets;% Actual validation labels/targets

% Calculate validation accuracy
val_correct_predictions = sum(val_predicted_classes == val_actual_classes);
val_total_samples = length(val_actual_classes);
val_accuracy = (val_correct_predictions / val_total_samples) * 100;

% Evaluate Optimization (Test) Accuracy
test_inputs = inputs(:, tr.testInd); % Test inputs
test_targets = targets(tr.testInd); % Test targets

test_outputs = net(test_inputs); % Get predictions
test_predicted_classes = round(test_outputs);
test_actual_classes = test_targets;

% Ensure all class labels are positive integers starting from 1
min_class = min([test_actual_classes, test_predicted_classes]); % Get the minimum class value
if min_class <= 0
    % Adjust the class labels to make them positive integers starting from 1
    test_actual_classes = test_actual_classes - min_class + 1;
    test_predicted_classes = test_predicted_classes - min_class + 1;
```

```
end

% Calculate optimization accuracy
test_correct_predictions = sum(test_predicted_classes == test_actual_classes);
test_total_samples = length(test_actual_classes);
test_accuracy = (test_correct_predictions / test_total_samples) * 100;

% Save the Optimized Model and Results
results.val_accuracy = val_accuracy;      % Validation accuracy
results.test_accuracy = test_accuracy;    % Test accuracy
results.num_components = num_components;  % Number of PCA components used
results.net = net;                        % Trained NN with PCA

save('optimized_trained_results.mat', 'results'); % Save the results

% Print Accuracy Results
fprintf('Validation Accuracy After PCA: %.2f%%\n', val_accuracy);
fprintf('Test (Optimization) Accuracy After PCA: %.2f%%\n', test_accuracy);

% Visualize PCA Results and NN Performance
figure;
pareto(explained); % Plot variance explained by each principal component
title('PCA Explained Variance');

figure;
plotperform(tr); % NN performance plot (training, validation, test errors)
```

```
% One-hot encode targets and predictions for confusion matrix
num_classes = max([test_actual_classes, test_predicted_classes]); % Total number of unique classes
test_actual_onehot = ind2vec(test_actual_classes, num_classes);    % One-hot encode actual classes
test_predicted_onehot = ind2vec(test_predicted_classes, num_classes); % One-hot encode predicted classes

% Plot confusion matrix with one-hot encoding
figure;
plotconfusion(test_actual_onehot, test_predicted_onehot);
```

7.4 4th code file Using Loop for training the data set

```
% Initialize an array to store results for each iteration
train_accuracies = zeros(10, 1); % Store training accuracies
test_accuracies = zeros(10, 1);  % Store test accuracies
all_results = cell(10, 1);       % Store results for each iteration

% Run the training and evaluation 10 times
for i = 1:10
    fprintf('Iteration %d of 10\n', i);

    % Load the merged dataset
    load('mergedData.mat');

    % Separate features and targets
    inputs = merged_data(:, 1:end-1)'; % Features (transpose to match NN input format)
    targets = merged_data(:, end)';    % User identifiers/targets (transpose for NN format)
```

```
% Normalize the input data
inputs = normalize(inputs, 'range'); % Normalize to [0, 1] range

% Create and configure the neural network
hiddenLayerSize = [20, 10]; % Define hidden lauer size
net = feedforwardnet(hiddenLayerSize, 'trainlm'); % Feedforward NN with Levenberg-Marquardt algorithm

% Divide data into training, validation, and test sets
net.divideParam.trainRatio = 0.7; % 70% training
net.divideParam.valRatio = 0.15;  % 15% validation
net.divideParam.testRatio = 0.15; % 15% testing

% Set training parameters
net.trainParam.epochs = 1000;  % Maximum number of epochs
net.trainParam.goal = 1e-6;    % Performance goal (MSE)
net.trainParam.min_grad = 1e-7; % Minimum gradient

% Train the neural network
[net, tr] = train(net, inputs, targets);

% Evaluate training accuracy
train_outputs = net(inputs(:, tr.trainInd)); % Network predictions for training set
train_predicted_classes = round(train_outputs); % Round predictions to nearest integer
train_actual_classes = targets(tr.trainInd);    % Actual training targets
```

```
% Calculate training accuracy
train_correct_predictions = sum(train_predicted_classes == train_actual_classes);
train_total_samples = length(train_actual_classes);
train_accuracy = (train_correct_predictions / train_total_samples) * 100;
train_accuracies(i) = train_accuracy; % Store training accuracy

% Evaluate test accuracy
test_inputs = inputs(:, tr.testInd); % Inputs for the test set
test_targets = targets(tr.testInd); % Targets for the test set

% Generate predictions for the test set
test_outputs = net(test_inputs);
test_predicted_classes = round(test_outputs);
test_actual_classes = test_targets;

% Calculate test accuracy
test_correct_predictions = sum(test_predicted_classes == test_actual_classes);
test_total_samples = length(test_actual_classes);
test_accuracy = (test_correct_predictions / test_total_samples) * 100;
test_accuracies(i) = test_accuracy; % Store test accuracy

% Store the results for each iteration
results.train_accuracy = train_accuracy;   % Training accuracy
results.test_accuracy = test_accuracy;     % Test accuracy
results.net = net;                         % Trained neural network
results.training_record = tr;              % Training record (training, validation, test performance)
```

```
    % Save the results for each iteration
    save(sprintf('trained_results_%d.mat', i), 'results');    % Save results for each iteration

    fprintf('Iteration %d: Training Accuracy = %.2f%%, Test Accuracy = %.2f%%\n', i,
train_accuracy, test_accuracy);
end

% Save all iteration results
save('all_results.mat', 'train_accuracies', 'test_accuracies', 'all_results');

% Print final message
fprintf('Training and testing completed for 10 iterations. Results saved.\n');
```

7.5 5th Code Optimize the loop trained datasets

```
% Load the dataset
load('mergedData.mat');

% Separate features and targets
features = merged_data(:, 1:end-1); % Features
targets = merged_data(:, end);     % Targets

% Handle multicollinearity by removing highly correlated features
corr_matrix = corrcoef(features); % Correlation matrix
% Find indices of redundant features (absolute correlation > 0.99)
redundant_columns = [];
for i = 1:size(corr_matrix, 1)
    for j = i+1:size(corr_matrix, 2)
```

```
        if abs(corr_matrix(i, j)) > 0.99
            redundant_columns = [redundant_columns, j]; % Collect the column indices
        end
    end
end
redundant_columns = unique(redundant_columns); % Remove duplicate indices

% Remove redundant features
if ~isempty(redundant_columns)
    fprintf('Removing %d redundant features due to high correlation.\n',
length(redundant_columns));
    features(:, redundant_columns) = []; % Remove redundant features
end

% Apply PCA to reduce dimensions
[coeff, pca_features, ~, ~, explained] = pca(features); % Perform PCA

% Retain enough components to explain 95% variance
explained_variance_threshold = 95;
cumulative_explained = cumsum(explained);
num_components = find(cumulative_explained >= explained_variance_threshold, 1);

% Select reduced features
optimized_features = pca_features(:, 1:num_components);

% Normalize the reduced features
optimized_features = normalize(optimized_features, 'range');
```

```
% Transpose data for neural network
inputs = optimized_features'; % Transpose features
targets = targets';           % Transpose targets

% Initialize arrays to store validation and test accuracy for multiple iterations
val_accuracies = zeros(10, 1);
test_accuracies = zeros(10, 1);
all_results = cell(10, 1); % Store results for each iteration

% Loop for training and testing 10 times
for iteration = 1:10
    fprintf('Iteration %d of 10\n', iteration);

    % Create and configure the neural network
    hiddenLayerSize = [20, 10]; % Hidden layer configuration
    net = feedforwardnet(hiddenLayerSize, 'trainlm'); % Create a new NN

    % Divide data into training, validation, and test sets
    net.divideParam.trainRatio = 0.7; % 70% training
    net.divideParam.valRatio = 0.15;  % 15% validation
    net.divideParam.testRatio = 0.15; % 15% testing

    % Set training parameters
    net.trainParam.epochs = 1000;  % Max epochs
    net.trainParam.goal = 1e-6;    % Performance goal
    net.trainParam.min_grad = 1e-7; % Minimum gradient
```

```
% Train the network with PCA-reduced inputs
[net, tr] = train(net, inputs, targets);

% Evaluate validation accuracy
val_inputs = inputs(:, tr.valInd); % Validation inputs
val_targets = targets(tr.valInd);  % Validation targets

val_outputs = net(val_inputs); % Get predictions
val_predicted_classes = round(val_outputs); % Round to nearest integer
val_actual_classes = val_targets;

% Calculate validation accuracy
val_correct_predictions = sum(val_predicted_classes == val_actual_classes);
val_total_samples = length(val_actual_classes);
val_accuracy = (val_correct_predictions / val_total_samples) * 100;
val_accuracies(iteration) = val_accuracy; % Store validation accuracy

% Evaluate test accuracy
test_inputs = inputs(:, tr.testInd); % Test inputs
test_targets = targets(tr.testInd); % Test targets

test_outputs = net(test_inputs); % Get predictions
test_predicted_classes = round(test_outputs);
test_actual_classes = test_targets;

% Calculate test accuracy
```

```
    test_correct_predictions = sum(test_predicted_classes == test_actual_classes);
    test_total_samples = length(test_actual_classes);
    test_accuracy = (test_correct_predictions / test_total_samples) * 100;
    test_accuracies(iteration) = test_accuracy; % Store test accuracy

    % Store the results for each iteration
    results.val_accuracy = val_accuracy;      % Validation accuracy
    results.test_accuracy = test_accuracy;    % Test accuracy
    results.num_components = num_components;  % Number of PCA components used
    results.net = net;                        % Trained NN with PCA
    results.training_record = tr;             % Training record

    all_results{iteration} = results;

    % Save the results for the current iteration
    save(sprintf('optimized_trained_results_%d.mat', iteration), 'results');

    fprintf('Iteration %d: Validation Accuracy = %.2f%%, Test Accuracy = %.2f%%\n', iteration,
val_accuracy, test_accuracy);
end

% Save all results
save('optimized_all_results.mat', 'val_accuracies', 'test_accuracies', 'all_results');

% Display summary results
fprintf('Average Validation Accuracy: %.2f%%\n', mean(val_accuracies));
fprintf('Average Test Accuracy: %.2f%%\n', mean(test_accuracies));
```

```
% Visualize PCA Explained Variance
figure;
pareto(explained); % Plot variance explained by each principal component
title('PCA Explained Variance');
```

7.6 Individual Workload files

Training all files Link: https://liveplymouthac-my.sharepoint.com/:f:/g/personal/10899162_students_plymouth_ac_uk/EuUcUGSnCS9CsJBzAIJKJdwBvKkuXLoZ4yMulGEjTNfBTA?e=0mHtYc