MTH209 Group 16 Project Report

200738 Pulkit Dhamija 208160499 Kaushik Raj Nadar
200890 Satmeet Singh Saluja 200314 Dev Bhavin Barbhaya
220481 Jiyanshu Dhaka

April 2024

1 Introduction

Electromyography is a technique for recording biomedical electrical signals obtained from the neuromuscular activities. Electromyograms (EMGs) have long been used to control prosthetic limbs, including hands and wrists. These signals, recorded from specific muscles associated with hand and finger gestures, can be harnessed to control various movements. However, classifying individual finger (IF) gestures is more challenging than whole-hand and wrist gestures due to the intricate muscle usage involved. Moreover, the response time of the gesture recognition system needs to be short enough to be perceived as real-time recognition by users.

Therefore, in this study, we developed a hand/finger gesture recognition model using only Time Domain (TD) features. We employed classification algorithms, namely Support Vector Machine (SVM), K-Nearest Neighbours (kNN), and logistic regression (LR), as classifiers, and their performances were statistically compared. A total of twelve gestures (including rest) were classified. We limited the number of channels used for recording EMG signals to reduce the complexity and improve the usability of the recognition system. Hence, two channels, which are relatively few compared to those used in previous studies, were used to record EMG signals from the muscles on a forearm. Six TD features were extracted from each channel, and therefore, a total of 12 TD features were used as input data. Kernel SVM was found to be the best performing model in our case.

2 Methods

2.1 Dataset

Our study utilizes electromyography (EMG) data, captured at a 100Hz sampling rate using dual-channel surface EMG (sEMG) sensors with Arduino microcontroller. Specifically, it captures two numerical values within the range of 0 to 1023. These values represent the intensity of muscle activity and are crucial for understanding muscle behavior. This dataset encompasses time series sEMG data for 12 distinct hand gestures including 'rest', 'index finger', '3 fingers', '4 fingers', 'thumb finger', 'okay', 'finger gun', 'scissors', 'middle finger', 'ring finger', 'little finger', and 'hand open'. This is particularly useful for multiclass-classification with the sEMG signal as the independent variable and corresponding hand gesture label as the dependent variable.

Data acquisition was performed using an advanced user interface (UI) featuring an automated timer. This timer alternates between a resting state and one of the 11 non-rest hand gestures every 5 seconds, guiding participants to mimic the displayed gesture. Each gesture is recorded over a span of 2 minutes. Therefore, each gesture has approximately 6,000 data points, and thus, the overall total data points of non-rest gestures are $11 \times 6,000 = 66,000$.

To ensure consistent data representation, a Min-Max Normalization technique was applied. This normalization process involves dividing the recorded data by 1023, effectively scaling it to fit within the 0 to 1 range. By doing so, the data maintains its relative proportions while becoming more amenable for further analysis.

Next, we performed segmentation on the time-varying data. This involved dividing the continuous stream of EMG readings into smaller windows of data. Each window captured a specific time interval, allowing for more focused analysis. These windows were then labeled with specific attributes, such as the type of muscle movement or gesture associated with that segment.

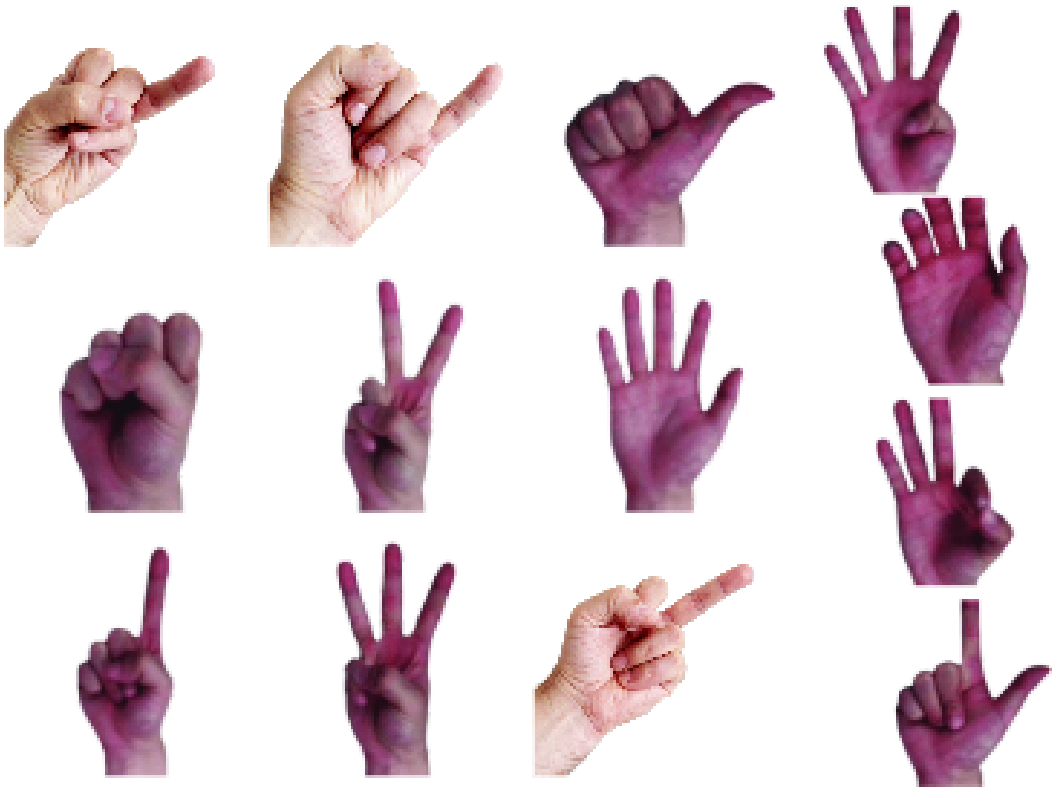

Figure 1: Hand Gestures used as Labels for the Classifier

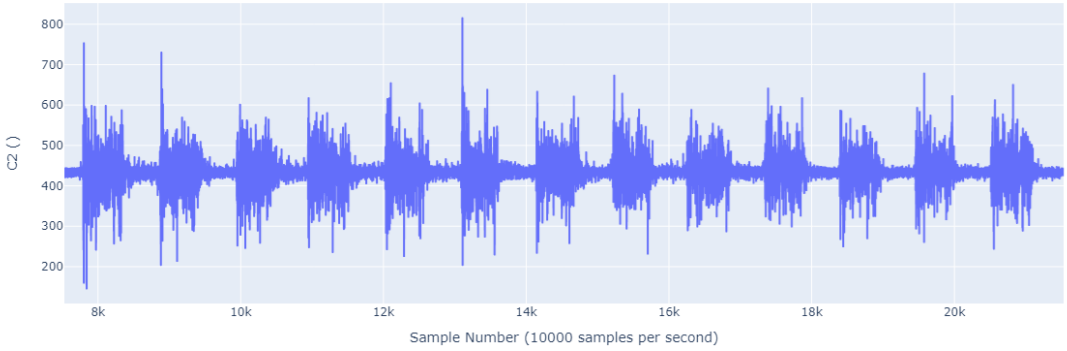

Figure 2: Raw sEMG Recording

Two critical hyperparameters, namely the window size (duration of each window) and the step size (how much the window shifts between adjacent segments), significantly influenced the final results. To optimize the classification performance, it was essential to explore various combinations of window sizes and step sizes during experimentation. In our case, we used a window size of 1000ms and step size of 10ms.

Assigning labels to each window was a crucial step. The majority label within a window's data determined its assigned gesture. For instance, if most of the data within a window corresponded to a specific hand gesture motion (e.g., index finger), that window would be labeled accordingly. Finally, the total number of samples available for fitting our model after cleaning, preprocessing, segmenting, and undersampling the data = 16302 (with window size=1000ms, step size=10ms).

2.2 Feature Extraction

In order for a gesture recognition system to be effective, its response time must be short enough to be perceived as real-time by users. We achieved this by utilizing Time Domain (TD) features as they are computationally efficient.

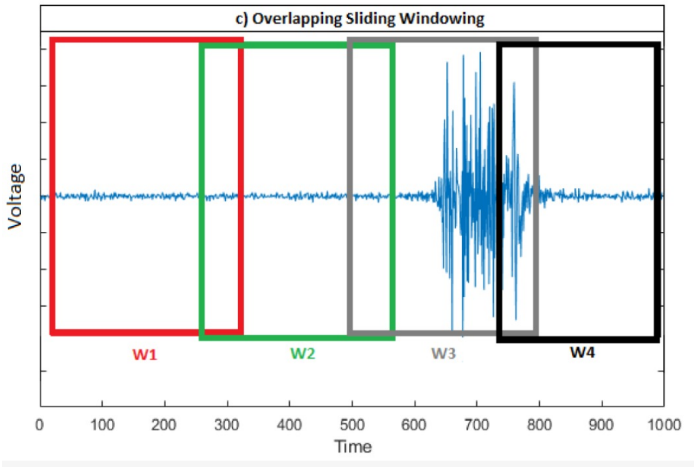

Figure 3: Segmentation of sEMG data

To extract meaningful information from the EMG (electromyography) signal, we employed several techniques:

- **Root Mean Square (RMS)**: The RMS quantifies the average magnitude of the EMG signal within a given window. It provides insights into muscle activation intensity. Mathematically, the RMS is calculated as:

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

  where x_i represents the EMG signal amplitude at each time point within the window, and N is the total number of samples.

- **Absolute Differential Signal/Waveform Length**: Capturing the cumulative length of the signal waveform, this feature reflects its complexity. It is calculated as the sum of absolute differences between consecutive samples:

$$\text{Waveform Length} = \sum_{i=1}^{N-1} |x_{i+1} - x_i|$$

- **Variance**: Variance measures the variability of the EMG signal within a window. It is computed as:

$$\text{Variance} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

  where μ is the mean of the signal values.

- **Slope Sign Change**: This feature examines how rapidly the signal changes direction, which can indicate muscle transitions. It counts the number of times the slope of the signal changes sign (from positive to negative or vice versa).

- **Mean Absolute Value (MAV)**: Representing the average absolute amplitude of the signal, MAV is calculated as:

$$\text{MAV} = \frac{1}{N}\sum_{i=1}^{N} |x_i|$$

- **Zero Crossing (ZC)**: the zero-crossing rate measures the number of times a signal crosses the zero level within a given segment. It's a simple way to assess the smoothness of a signal. For a signal x_h at window k, the zero-crossing rate (ZC_k) is calculated as:

$$ZC_k = \sum_{h=k}^{M+k-1} |\text{sign}(x_h) - \text{sign}(x_{h-1})|$$

  where:
  M is the step between analysis windows.
  N is the analysis window length.

2.3 Classification Models

In this section, we explore multiclass classification using various algorithms: **SVM**, **Logistic Regression** and **KNN**.
SVM is chosen for its ability to effectively handle high-dimensional data and find optimal decision boundaries. Logistic Regression, on the other hand, provides a probabilistic framework suitable for multiclass classification tasks. We will compare their performances to determine the most suitable model.

Support Vector Machine (SVM) for Multiclass Classification

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for both binary and multiclass classification tasks. In the context of multiclass classification, SVM aims to find the optimal hyperplane that best separates the data into multiple classes.

Optimal Hyperplane:
SVM seeks to find the hyperplane that maximizes the margin between different classes while minimizing the classification error. The hyperplane is defined as:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

where: (w) is the weight vector, (x) is the input feature vector, (b) is the bias term.

Margin:
The margin is the distance between the hyperplane and the nearest data point from either class. SVM aims to maximize this margin, as it provides better generalization and reduces overfitting.

Soft Margin:
In cases where the data is not linearly separable, SVM introduces a slack variable $(\xi_i))$ to allow for misclassification. The objective function is then modified to include a penalty for misclassified points:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

where (C) is the regularization parameter, and $(\xi_i))$ is the slack variable.

Kernel Trick:
SVM can efficiently handle nonlinearly separable data by mapping the input features into a higher-dimensional space using a kernel function (ϕ). This allows for linear separation in the transformed feature space without explicitly computing the transformed vectors. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

Optimization:
SVM solves for the parameters (w) and (b) by solving the constrained optimization problem using techniques like quadratic programming or gradient descent. The optimization problem is typically expressed in the form of a Lagrangian:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N}\alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1)$$

where is the (α) vector of Lagrange multipliers, and (y_i) is the class label of the i-th sample.
The second term in the optimization equation is a part of the Lagrangian function in the context of constrained optimization. In the case of SVM, it represents the sum of individual losses incurred by misclassified data points, also known as the hinge loss.

$$\sum_{i=1}^{N}\alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1)$$

N is the total number of data points, α_i is the Lagrange multiplier associated with the i-th data point, y_i is the class label of the i-th data point, w and b are the parameters of the hyperplane, $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1$ represents the decision function output for the i-th data point. If this value is greater than 1, the data point is correctly classified. If it's less than 1, the data point is misclassified.

The term $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1$ computes the loss incurred by each data point. When this value is

positive (i.e., the data point is outside the margin but still on the correct side of the hyperplane), it contributes to the overall loss. Otherwise, if the value is negative (i.e., the data point is misclassified or within the margin), the loss is zero.

The Radial Basis Function (RBF) is a kernel function used in Support Vector Machines (SVM) for handling non-linear classification problems. It allows SVM to map the input data into a higher-dimensional space where it can be linearly separable, enabling SVM to find non-linear decision boundaries.

The RBF kernel function is defined as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

where: $K(\mathbf{x}_i, \mathbf{x}_j)$ is the RBF kernel function value between input feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, $|\mathbf{x}_i - \mathbf{x}_j\|$ is the Euclidean distance between the input feature vectors, σ is a parameter known as the bandwidth or width of the Gaussian distribution.

How RBF is Used in SVM:

In SVM, the decision boundary is determined by a hyperplane in the feature space. The RBF kernel allows SVM to implicitly map the input features into a higher-dimensional space where the data might become linearly separable. This is achieved by computing the dot product of the feature vectors in the higher-dimensional space without explicitly calculating the transformed feature vectors, thanks to the kernel trick.

Decision Function:
The decision function of the SVM using the RBF kernel is given by:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

where $\mathbf{x}$ is the decision function for a new data point $\mathbf{x}$, α_i are the Lagrange multipliers obtained during training, y_i are the class labels of the training data, b is the bias term, N is the number of support vectors.

Gradient Descent Algorithm for Constrained Optimization in SVM

In SVM, the optimization problem involves finding the parameters ($\mathbf{w}$ and b) and of the hyperplane that maximizes the margin between different classes while minimizing the classification error. The constrained optimization problem is typically expressed in the form of a Lagrangian, which incorporates the constraints as penalty terms using Lagrange multipliers.

Gradient Descent Algorithm Steps:

1. Initialization: Start with initial values for the parameters ($\mathbf{w}$ and b) and choose a learning rate η.
2. Calculate the Gradient: Compute the gradient of the Lagrangian with respect to the parameters ($\mathbf{w}$ and b). The gradient represents the direction of steepest ascent.
3. Update Parameters: Update the parameters using the gradient and the learning rate. This step involves moving towards the direction of the negative gradient to minimize the objective function.
4. Projection: Project the updated parameters onto the feasible region to satisfy the constraints of the optimization problem. In SVM, this typically involves adjusting the parameters to maintain the margin and ensure correct classification.
5. Convergence Check: Repeat steps 2-4 until convergence criteria are met, such as reaching a predefined number of iterations or achieving a sufficiently small change in the objective function.

Gradient Calculation: The gradient of the Lagrangian with respect to the parameters ($\mathbf{w}$ and b) is computed as follows:

$$\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i$$

where $\mathbf{x}_i$ is the feature vector of the i-th data point, y_i is the class label, and α_i is the Lagrange multiplier.

Parameter Update: Update the parameters ($\mathbf{w}$ and b) sing the gradient and the learning rate (eta:

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \nabla_{\mathbf{w}} \mathcal{L}$$

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \nabla_{\mathbf{w}} \mathcal{L}$$

Projection: Adjust the parameters to satisfy the constraints of the optimization problem, ensuring the margin and correct classification.

Logistic Regression for Multiclass Classification

Logistic Regression is a popular supervised learning algorithm used for binary classification tasks. However, it can also be extended to handle multiclass classification by using techniques like one-vs-rest (OvR) or multinomial logistic regression. In logistic regression, the algorithm models the probability that a given input belongs to a particular class.

Regression Equation: In the case of multiclass classification, let's consider K classes (where $K > 2$). The logistic regression model predicts the probability of each class. For each class k, the probability $P(y = k|\mathbf{x})$ is computed using the softmax function:

$$P(y = k|\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x} + b_j}}$$

where $\mathbf{x}$ is the input feature vector, $\mathbf{w}_k$ and b_k are the weight vector and bias term for class k, e is the base of the natural logarithm.

Model Parameters:
The parameters $\mathbf{w}_k$ and b_k are learned from the training data using optimization techniques like gradient descent. The objective is to minimize the cross-entropy loss function:

$$J(\mathbf{w}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log(P(y = k|\mathbf{x}_i))$$

where N is the number of samples, $y_{i,k}$ is the indicator variable that equals 1 if the i-th sample belongs to class and 0 otherwise.

Assumptions:
Linearity: The relationship between the input features and the log-odds of the response variable is linear.
No Multicollinearity: The input features are assumed to be independent of each other.
Large Sample Size: Logistic Regression performs well with large sample sizes.

Probabilities:
After training the logistic regression model, the predicted probabilities for each class are computed using the softmax function. These probabilities indicate the likelihood of a given sample belonging to each class, allowing for multiclass classification.

Gradient Descent Algorithm for Constrained Optimization in Logistic Regression

Alogorithm is similar as in for SVM, just the objective function is now the Cross Entropy Loss function.

K-Nearest Neighbors (KNN) for Multiclass Classification

K-Nearest Neighbors (KNN) is a simple yet powerful supervised learning algorithm used for both classification and regression tasks. In the context of multiclass classification, KNN predicts the class of a new data point by considering the classes of its k nearest neighbours.

Algorithm Overview:

Training Phase:
In the training phase, KNN stores the feature vectors and corresponding class labels of the training data. No explicit training or model fitting is performed in KNN.

Prediction Phase:
In the prediction phase, when given a new data point, KNN calculates the distance between the new point and every point in the training set using a distance metric such as Euclidean distance, although other metrics such as Manhattan distance or cosine similarity can also be used.

K Nearest Neighbours:
KNN selects the k nearest neighbours of the new data point based on the calculated distances. The class labels of these k neighbours are examined.

Voting:
For multiclass classification, KNN uses a majority voting scheme to determine the class label of the new data point. The class label that appears most frequently among the k nearest neighbours is assigned to the new point.

Decision Boundary:
The decision boundary in KNN is non-linear and depends on the distribution of the training data. KNN makes predictions based on the local structure of the data.

Distance Calculation: KNN typically uses the Euclidean distance to measure the distance between two data points x_i and x_j :

$$\text{Euclidean distance} = \sqrt{\sum_{l=1}^{d}(x_{i,l} - x_{j,l})^2}$$

where d is the number of dimensions (features) in the data.

Majority Voting: After identifying the k nearest neighbours, KNN assigns the class label of the new data point based on the majority class among these neighbours. Mathematically, it can be represented as:

$$\hat{y} = \text{argmax}_c \sum_{i=1}^{k} \delta(y_i, c)$$

where $\hat{y}$ is the predicted class label, c represents each class, and $\delta(y_i, c)$ is the Kronecker delta function, which returns 1 if y_i (the class label of the i-th neighbour) is equal to c and 0 otherwise.

3 Analysis and Results

3.1 Analysis

After extracting the features, the structure of the dataset can be seen in Figure 4.

label	Channel1_rms	Channel1_MAV	Channel1_WL	Channel1_SSC	Channel1_VAR	Channel1_ZC	Channel2_rms	Channel2_MAV	Channel2_WL	Channel2_SSC	Channel2_VAR	Channel2_ZC
'Index+'	0.426696	10.581623	1.462366	21.0	0.002917	0.0	0.428490	10.703812	0.484848	15.0	0.000289	0.0
'Index+'	0.432446	10.725318	1.405670	21.0	0.002957	0.0	0.428767	10.710655	0.498534	16.0	0.000292	0.0
'Index+'	0.438917	10.903226	1.313783	19.0	0.002439	0.0	0.427539	10.678397	0.515152	17.0	0.000345	0.0
'Index+'	0.441732	10.977517	1.181818	18.0	0.002318	0.0	0.425147	10.618768	0.556207	17.0	0.000337	0.0

Figure 4: Dataset

To visualize potential clusters, we created a scatter plot of the dataset, focusing on two comparable features from the two channels. For eg. Channel1_MAV and Channel2_MAV
Figures 5, 6 and 7 show the scatter plot against the waveform length, mean absolute value, and slope sign change features of the two channels, respectively.

Further, we conducted principal component analysis (**PCA**) to identify the top two and top three principal components. Subsequently, we generated scatter plots using these new component features. Figures 8 and 9 show the scatter plots against the top two principal components and the top three principal components, respectively.

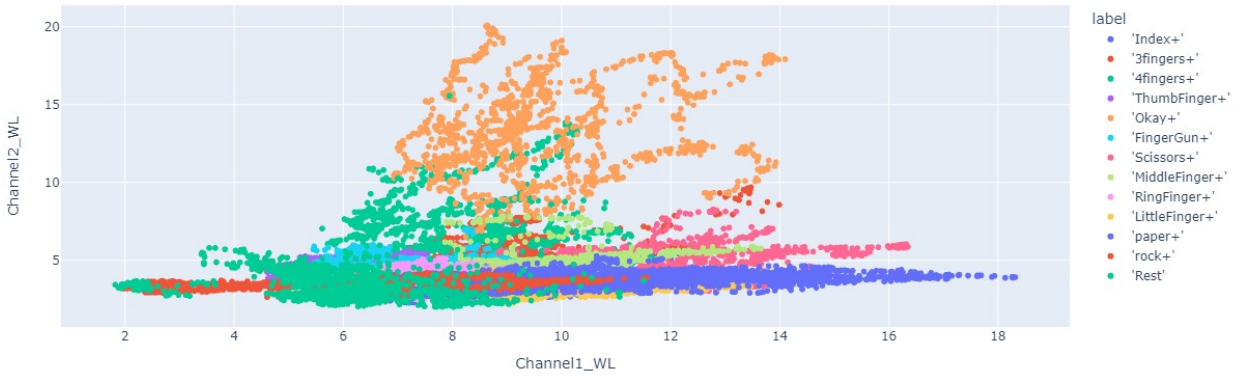

Figure 5: Scatter plot for waveform length

Figure 6: Scatter plot for mean absolute value

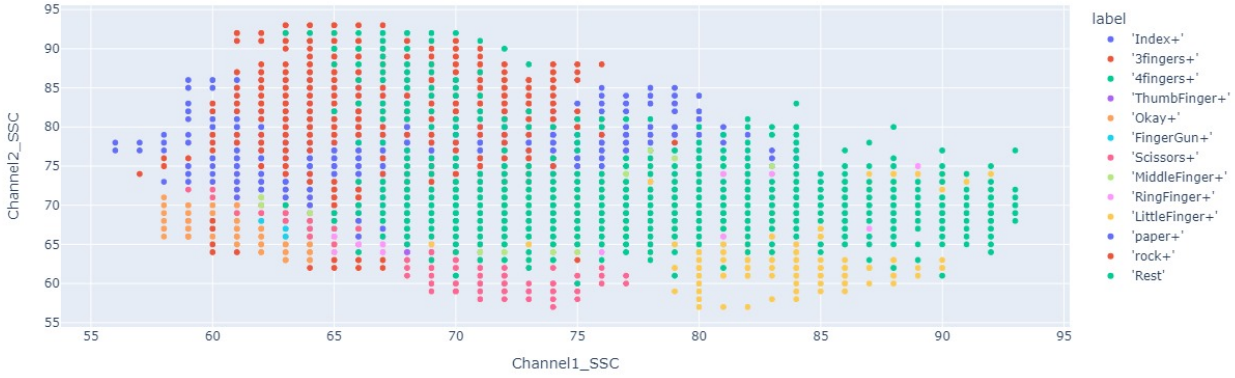

Figure 7: Scatter plot for slope sign change

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a popular dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving most of the variance in the data. It achieves this by identifying the directions (principal components) along which the data varies the most.
Steps of PCA:

1. Standardization: Standardize the features of the dataset to have a mean of 0 and a standard deviation of 1. This step ensures that all features contribute equally to the analysis.

2. Compute Covariance Matrix: Compute the covariance matrix of the standardized data. The covariance matrix captures the relationships between pairs of features and provides information about the variability and orientation of the data.

3. Eigenvalue Decomposition: Perform eigenvalue decomposition on the covariance matrix to obtain its eigenvectors and eigenvalues. The eigenvectors represent the directions (principal components) along which the data varies the most, and the eigenvalues indicate the amount of variance explained by each principal component

4. Select Principal Components: Sort the eigenvectors based on their corresponding eigenvalues in descending order. The eigenvectors with the highest eigenvalues represent the principal components that capture the most variance in the data. Choose the top k eigenvectors to form the new lower-dimensional space, where k is the desired dimensionality of the reduced space.

5. Projection: Project the original data onto the selected principal components to obtain the reduced-dimensional representation of the data.

Covariance Matrix: The covariance matrix C of the standardized data X with n samples and d features is calculated as:

$$C = \frac{1}{n-1}(\mathbf{X} - \bar{\mathbf{x}})^T(\mathbf{X} - \bar{\mathbf{x}})$$

where $\bar{\mathbf{x}}$ is the mean vector of the standardized data.

Eigenvalue Decomposition: The eigenvalue decomposition of the covariance matrix yields the eigenvectors $\mathbf{v}_i$ and eigenvalues λ_i, such that:

$$C\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

Projection: The original data $\mathbf{X}$ is projected onto the selected principal components $\mathbf{V}_k$ to obtain the reduced-dimensional representation $\mathbf{Y}$ as:

$$\mathbf{Y} = \mathbf{X}\mathbf{V}_k$$

where $\mathbf{V}_k$ contains the top k eigenvectors as columns

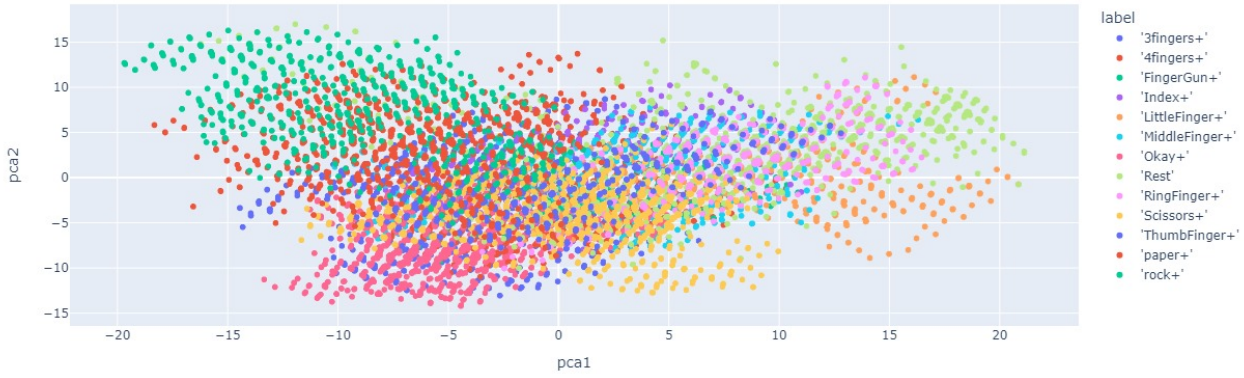

Figure 8: Scatter plot for top two principal components

Explained variance ratios for the top 3 components are 0.53959409, 0.31841828, and 0.07193782.

From these plots, we can infer that there is potential cluster formation within the dataset.

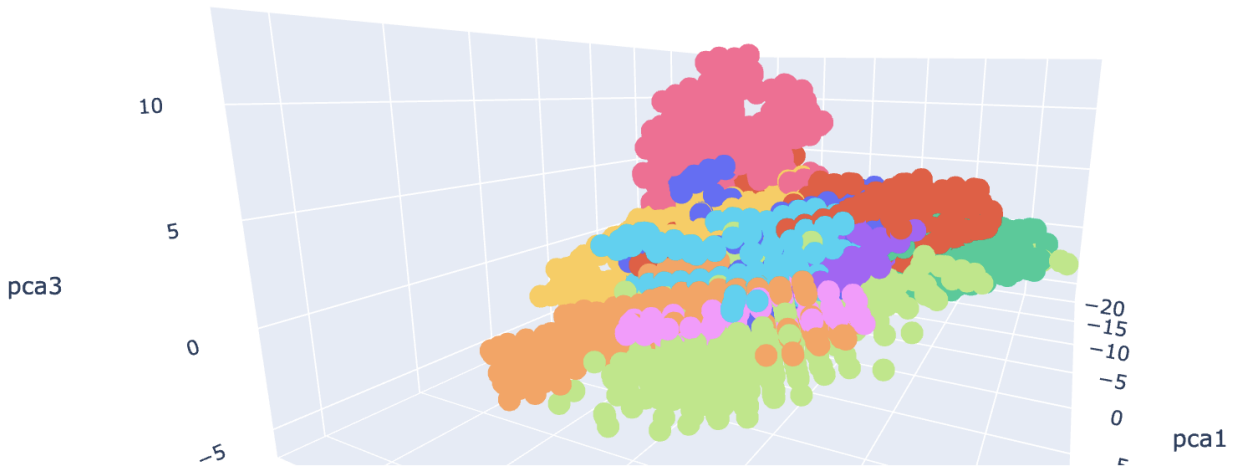

Figure 9: Scatter plot for top three principal components

3.2 Results

We perform a train-test split of the dataset, allocating 30% of the data for testing purposes and reserving the remaining 70% for training the model.

label code	0	1	2	3	4	5	6	7	8	9	10	11	12
Gestures	3 Fingers	4 Fingers	Finger Gun	Index	Little Finger	Middle Finger	Okay	Rest	Ring Finger	Scissors	Thumb Finger	Paper	Rock

Logistic Regression Results

Parameter values: Regularisation Parameter C=0.1
Training accuracy: 60.74%
Testing accuracy: 60.33%

Figure 10 shows the confusion matrix for testing dataset:

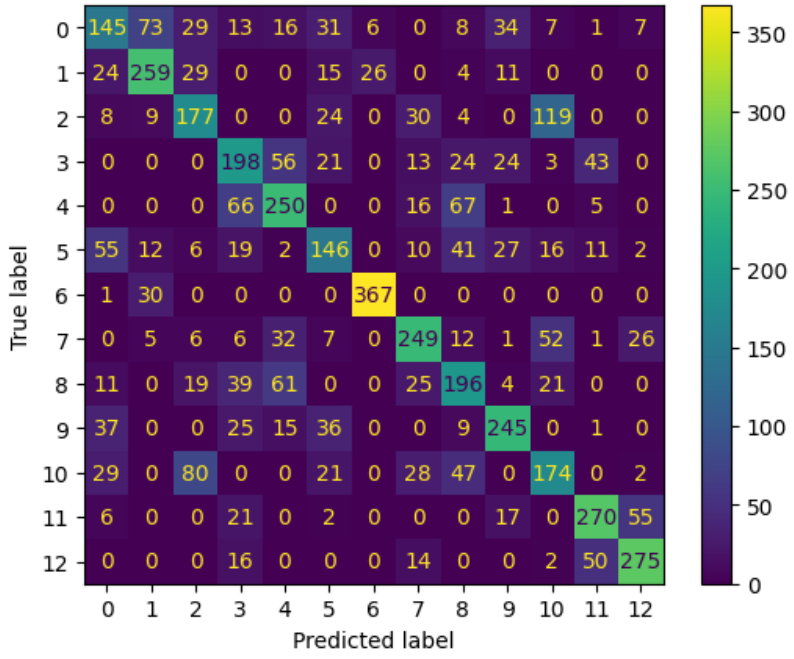

Figure 10: Confusion Matrix for test data

K-Neighbour Nearest

Hyperparameter values: K=1
Training accuracy: 100%
Testing accuracy: 92.78%

Figure 11 shows the confusion matrix for testing dataset:

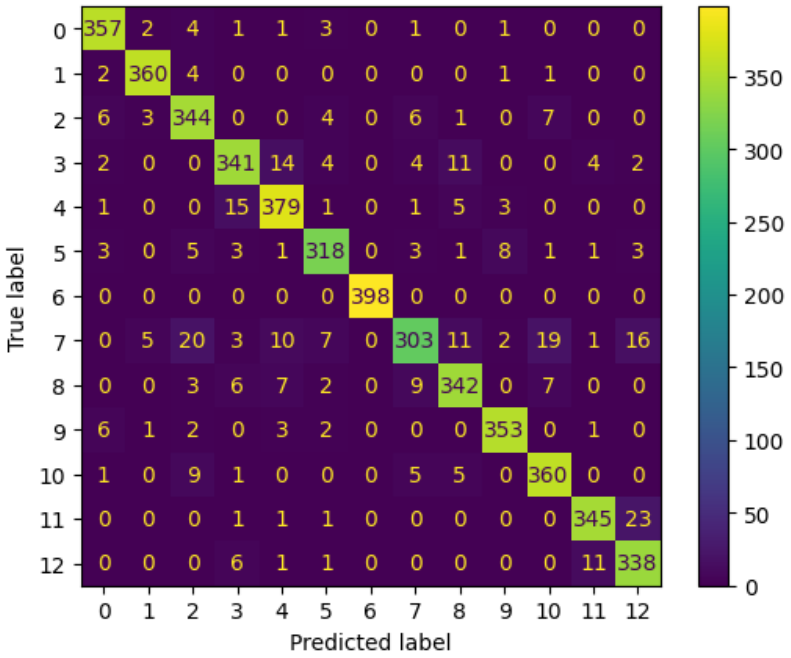

Figure 11: Confusion matrix for test data

Support Vector Machine

Radial Basis Function (RBF) kernel was used in this case to fit non-linear data.
Hyperparameter values: Regularisation Parameter C=20, γ=1
Training accuracy: 99.89%
Testing accuracy: 94.17%

Figure 12 shows the confusion matrix for testing dataset:

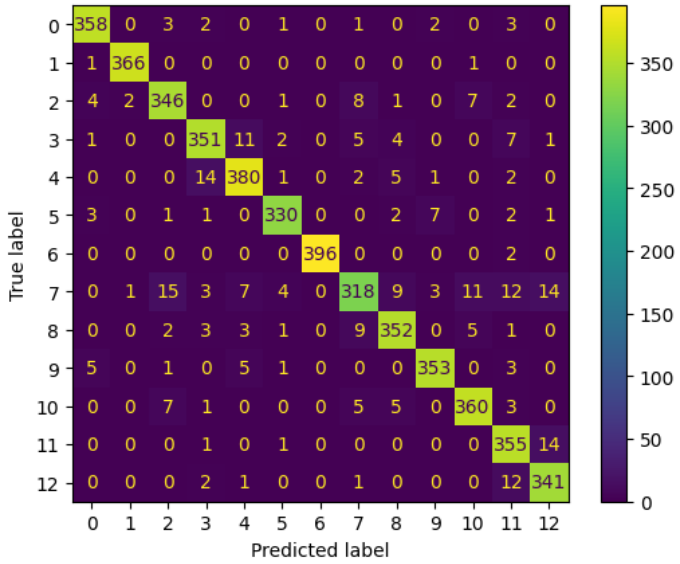

Figure 12: Confusion matrix for test data

4 Conclusion

Our project successfully analyzed EMG data using preprocessing, feature extraction, visualization, and classification techniques. The initial step involved preprocessing the electromyography (EMG) data. This process ensures that the data is clean, consistent, and ready for analysis. The EMG data was segmented to extract relevant portions for further investigation. From the two-channel sensors, six time domain features were extracted. These features serve as essential inputs for subsequent analysis.

To gain insights and identify patterns, the data was visualized using scatter plots. Scatter plots allow us to observe relationships between variables and detect any discernible clusters or trends. Principal Component Analysis (PCA) was employed to reduce the dimensionality of the data. By projecting the data onto a lower-dimensional space, we aimed to capture the most significant variations while minimizing information loss. The resulting reduced feature set was then visualized using scatter plots, providing a concise representation of the data. Our goal was to classify or predict outcomes based on the EMG data.

We explored several classification models, including: Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Logistic Regression. These models were trained on labeled data to learn patterns and make predictions. After training the models, we evaluated their performance. Among the three models, the SVM demonstrated the best performance in our specific case. SVM's ability to handle non-linear relationships and find optimal decision boundaries made it well-suited for our EMG classification task. The findings highlight the effectiveness of SVM for accurate classification. These insights can inform future applications in areas such as muscle movement recognition, prosthetics, and rehabilitation.