CHEMIG - PulseSort

* Tanmay Jain
* 241090

# Table Of Content:

1. Abstract
2. Introduction
3. Theory
4. Approach
5. Result
6. Observation
7. Conclusion

# Abstract

This project explores the classification of hand gestures using EMG (Electromyography) signals captured from muscle activity. Leveraging time-domain signal features—particularly Slope Sign Changes (SSC)—and three supervised machine learning models (Logistic Regression, Support Vector Machine, and K-Nearest Neighbours), we demonstrate gesture recognition with high accuracy. Preprocessing steps like normalization and segmentation, combined with robust feature extraction, yield a clean dataset suitable for model training. Among the features tested, SSC proved critical in improving classification performance. The models were implemented using scikit-learn, and comparative evaluation shows Logistic Regression and KNN achieving perfect accuracy, while SVM reached 75%. This work lays the foundation for intuitive human-computer interaction through muscle-based gesture recognition.

# Introduction

Interpreting human intent through muscle activity opens new frontiers for gesture-controlled interfaces and assistive technologies. This project, *PulseSort*, focuses on classifying hand gestures using EMG (Electromyography) signals—captured from forearm muscle contractions—translated into a structured dataset via an Arduino microcontroller. The goal is to develop a machine learning pipeline that can reliably distinguish between 12 unique hand gestures, enabling intuitive human-computer interaction.

To achieve this, we implement three supervised machine learning algorithms—Logistic Regression, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN)—and evaluate their performance on feature-engineered EMG signals. Key time-domain features such as RMS, Mean Square, Waveform Length, and especially Slope Sign Changes (SSC) are extracted to characterize gesture-specific patterns. The dataset undergoes normalization, segmentation, and encoding to facilitate robust model training using scikit-learn. Evaluation through precision, recall, and F1-scores reveals that accurate classification hinges not just on the choice of algorithm, but on insightful preprocessing and meaningful feature selection.

# Theory

## ML models used:

Three types of machine learning models that were trained for the task which included:

* **Logistic Regression:**

Logistic Regression is a **supervised learning algorithm** used primarily for **classification task.** Instead of predicting a continuous value like in linear regression, logistic regression predicts the **probability** that a given input belongs to a particular class. It uses the **sigmoid function** to squash outputs between 0 and 1: **σ(z) = 1 / (1 + e^(-z)), where z = wᵀx + b.** The output is interpreted as the probability of the positive class (usually labelled as 1). **Multinomial** logistic regression model has been used which classifies More than two unordered categories (e.g., cat/dog/sheep). Logistic regression uses **Maximum Likelihood Estimation (MLE)** to find the best weights. Tooles used are **scikit-learn**: LogisticRegression and **Confusion Matrix** for evaluation.

* **Support Vector Machine:**

Support Vector Machine (SVM) is a **supervised learning algorithm** used for both classification and regression tasks, though it's most famous for classification. It defines **hyperplane -** the decision boundary that separates different classes, **Support Vectors** - Data points closest to the hyperplane, **Margin** - The distance between the hyperplane and the support vectors. SVM aims to **maximize this margin** for better generalization. For **linearly separable data**, SVM finds the optimal hyperplane and for **non-linear data**, it uses the **kernel trick** to map data into higher dimensions where it becomes linearly separable. Kernel used here is RBF (Gaussian) which handles complex, non-linear relationships. Tools used are **scikit-learn** – SVC and confusion matrix for classification.

* **K – Nearest Neighbor**

K-Nearest Neighbours (KNN) is a **supervised learning algorithm** used for both classification and regression, though it's most applied to classification tasks. It’s intuitive, non-parametric, and doesn’t require a training phase—making it a classic “lazy learner.” KNN predicts the label of a new data point by looking at the **‘k’ closest points** in the training data using **majority voting** for classification and **averaging** for regression. To find the “nearest” neighbours, KNN uses distance measures like **Metric**, **Euclidean**, **Manhattan**, **Minkowski**, **Hamming.** Toolsused are KNeighborsClassifier from **scikit-learn and** StandardScaler for feature scaling.

## Time Domain Features:

From the EMG signals received from the sensors in two channels, the following time domain features are extracted for both the channels:

* **RMS:** Root mean square of the channel values.
* **MS:** Mean square of the channel values.
* **MAV:** Mean absolute value of the channels
* **WL:** Waveform length is sum of absolute difference between consecutive channel values
* **SSC (Most Important):** Slope Sign Changes (SSC) is specially used for analysing EMG signals. It helps quantify the **dynamic behaviour** of a signal by counting how many times the slope of the waveform changes direction—essentially capturing the signal’s complexity and frequency content.

# Approach

## Signal Capturing:

EMG sensors (which consists of electrodes and amplifier circuits) are attached to the muscles to capture the EMG signals. The sensor is then connected to a microcontroller (here Arduino is used, others like Raspberry PI can also be used) to read and quantify the analog signals given by the sensor into numbers as channel 1 and 2.

Arduino code for signal capture:

void setup() {

  Serial.begin(9600);

}

void loop() {

  int ch1 = analogRead(A0);  // EMG Channel 1

  int ch2 = analogRead(A1);  // EMG Channel 2

  Serial.print(ch1);

  Serial.print(",");

  Serial.print(ch2);

  Serial.print(",");

  Serial.println(millis());  // Time in ms

  delay(10);  // 100 Hz sampling rate

}

In this way a dataset is created which has 12 .csv files having EMG signal data as ch1, ch2 for the 12 gestures to be classified with labels.

## Preprocessing:

Before feeding this dataset to a machine learning model for training we first preprocess the dataset since it is a continuous analog signal containing noise. It may also have a poor signal to noise ratio; random data points recorded due to error or sensor defect. It also prepares the data for feature extraction.

Steps of pre-processing used:

* **Normalisation:** Scaling down of all the channel values to a continuous range of 0 to 1.
* **Segmentation:** Moving a window of a specified size over the dataset with a specified step size so that features can be extracted over each window.

**The column for label remains undisturbed.**

## Feature Extraction:

Feature extraction is the process of transforming raw data into a set of **meaningful, informative attributes**—called features—that can be used for analysis, modelling, or classification.

**Why Feature Extraction Is Important?**

| **Benefit** | **Description** |
| --- | --- |
| **Dimensionality Reduction** | Simplifies complex data by reducing the number of variables |
| **Improved Model Accuracy** | Helps algorithms focus on the most relevant patterns |
| **Faster Computation** | Reduces training time and memory usage |
| **Noise Reduction** | Filters out irrelevant or redundant information |
| **Better Generalization** | Prevents overfitting by focusing on core signal characteristics |
| **Interpretability** | Makes models easier to understand and debug |

Features like RMS, MS, MAV, WL and SSC are extracted on each segmentation window for each gesture to create its feature file. The feature files for all the gesture are concatenated to create the final labelled dataset.

**The column for label remains undisturbed.**

## Data Splitting:

The feature extracted dataset is then split into training and testing data in the ratio of 80-20 so that the trained model can be evaluated on the 20% unseen test data.

## Training:

Training data is then fed to the ML model for weight and bias training without label, one by one for all three types of models – **Logistic Regression**, **Support Vector Machine** and **K – Nearest Neighbour** by using predefined classes of **scikit-learn** library.

## Evaluation:

The three models are then used to predict the label of the test data, and the prediction is compared to the label of test data for evaluation.

# Source Code

1. # %% [markdown]
2. # # Feature extraction code of Task 2
3. # %%
4. import numpy as np
5. import pandas as pd
6. def normalize(df):
7. from sklearn.preprocessing import MinMaxScaler
8. scaler=MinMaxScaler();
10. norm\_df=pd.DataFrame(scaler.fit\_transform(df[['ch1','ch2']]))
11. norm\_df.columns=['ch1','ch2']
12. norm\_df['time']=df['timestamp\_ms']
13. return norm\_df
14. def mean\_sq(arr):
15. ms=0
16. for i in arr:
17. ms+=(i\*\*2)
18. return ms/len(arr)
19. def compute\_ssc(signal):
20. diff1 = np.diff(signal[:-1])
21. diff2 = np.diff(signal[1:])
22. # return np.sum(((diff1 \* diff2) < 0) & (np.abs(diff1 - diff2) > threshold))
23. return np.sum((diff1 \* diff2) < 0)
24. def extract(df,window\_size=100,step\_size=10):
25. new\_df=pd.DataFrame({
26. 'time':[],
27. 'rms\_ch1':[],
28. 'rms\_ch2':[],
29. 'mav\_ch1':[],
30. 'mav\_ch2':[],
31. 'mse\_ch1':[],
32. 'mse\_ch2':[],
33. 'WL\_ch1':[],
34. 'WL\_ch2':[],
35. 'SSC\_ch1':[],
36. 'SSC\_ch2':[],
37. })
38. n=len(df)
39. for i in range(0,n-window\_size+1,step\_size):
40. ch1=df['ch1'][i:i+window\_size] # to access only 100 elements of ch1 at a time
41. ch2=df['ch2'][i:i+window\_size]
43. row={
44. 'mse\_ch1':[mean\_sq(ch1)],
45. 'mse\_ch2':[mean\_sq(ch2)],
46. 'mav\_ch1':[np.mean(ch1)],
47. 'mav\_ch2':[np.mean(ch2)],
48. 'time':[np.mean(df['time'][i:i+window\_size])],
49. 'WL\_ch1':[np.sum(np.abs(np.diff(ch1)))],
50. 'WL\_ch2':[np.sum(np.abs(np.diff(ch2)))],
51. 'SSC\_ch1':[compute\_ssc(df['ch1'])],
52. 'SSC\_ch2':[compute\_ssc(df['ch2'])],
53. }
55. row['rms\_ch1']=np.sqrt(row['mse\_ch1'])
56. row['rms\_ch2']=np.sqrt(row['mse\_ch2'])
58. new\_df=pd.concat([new\_df,pd.DataFrame(row)],ignore\_index=True)
60. return new\_df
61. def get\_feature(file\_loc,suffix):
62. df=pd.read\_csv(file\_loc)
63. norm\_df=normalize(df)
64. new\_df=extract(norm\_df)
65. new\_df['label']=np.full(len(new\_df),df['label'][0])
66. new\_df.to\_csv(f'features\_{suffix}.csv',index=False)
67. return new\_df
68. # %% [markdown]
69. # # 1. Combine all Feature Files
70. # %%
71. files=['clenched\_data.csv','fist\_data.csv','four\_data.csv','index\_finger\_data.csv','okay\_data.csv','peace\_data.csv','rest\_data.csv','rock\_data.csv','spread\_data.csv','three\_data.csv','thumb\_data.csv','up\_data.csv']
72. df=pd.DataFrame()
73. for f in files:
74. df=pd.concat([df,get\_feature(f'Dataset/{f}',f[0:f.rindex('\_')])],ignore\_index=True)
75. df
76. # %% [markdown]
77. # #  2. Encode the Labels
78. # %% [markdown]
79. # - Use LabelEncoder from sklearn.preprocessing to convert gesture names into numeric labels (0–11)
80. # %%
81. from sklearn.preprocessing import LabelEncoder
82. enc=LabelEncoder()
83. df['label']=enc.fit\_transform(df['label'])
84. df
85. # %% [markdown]
86. # - Print a dictionary mapping of label numbers to gesture names.
87. #
88. # %%
89. enc.classes\_ # converted to numeric label from 0 - 11
90. # %%
91. # Method - 1
92. print(pd.concat([pd.DataFrame(enc.classes\_),pd.DataFrame(df['label'].unique())],axis='columns'))
93. # %%
94. # Method - 2
95. label\_to\_gesture={i:label for i,label in enumerate(enc.classes\_)}
96. print(label\_to\_gesture)
97. # %% [markdown]
98. # # 3. Split the Dataset
99. # %%
100. from sklearn.model\_selection import train\_test\_split
101. x=df.drop(columns=['time','label'])
102. # x=df[['SSC\_ch1','SSC\_ch2']] for 100% accuracy on all models
103. x
104. # %%
105. y=df['label']
106. x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,test\_size=0.2,stratify=y)
107. # %%
108. x\_train , y\_train
109. # %% [markdown]
110. # # 4-6. Training, evaluation and visualisation of Logistic Regression, KNN and SVM Models
111. # %%
112. from sklearn.linear\_model import LogisticRegression
113. from sklearn.svm import SVC
114. from sklearn.neighbors import KNeighborsClassifier
115. from sklearn.metrics import classification\_report, confusion\_matrix
116. import seaborn as sns
117. import matplotlib.pyplot as plt
118. models = {
119. "Logistic": LogisticRegression(multi\_class='multinomial',solver='lbfgs',max\_iter=1000),
120. "SVM": SVC(kernel='rbf',max\_iter=1000),
121. "KNN": KNeighborsClassifier(n\_neighbors=5)
122. }
123. for name, model in models.items():
124. model.fit(x\_train, y\_train)
125. preds = model.predict(x\_test)
126. print(f"{name} Report:\n", classification\_report(y\_test, preds))
127. cm=confusion\_matrix(y\_test, preds)
128. sns.heatmap(cm,annot=True,fmt='d',cmap='YlGnBu')
129. plt.title(f"{name} Confusion Matrix")
130. plt.ylabel("Actual Class")
131. plt.xlabel("Predicted Class")
132. plt.show()

# Results

The classification report and confusion matrix for all three models is given below:

**Logistic Regression**

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 1.00 | 1.00 | 1.00 | 118 |
| 1 | 1.00 | 1.00 | 1.00 | 118 |
| 2 | 1.00 | 1.00 | 1.00 | 119 |
| 3 | 1.00 | 1.00 | 1.00 | 118 |
| 4 | 1.00 | 1.00 | 1.00 | 118 |
| 5 | 1.00 | 1.00 | 1.00 | 118 |
| 6 | 1.00 | 1.00 | 1.00 | 118 |
| 7 | 1.00 | 1.00 | 1.00 | 118 |
| 8 | 1.00 | 1.00 | 1.00 | 119 |
| 9 | 1.00 | 1.00 | 1.00 | 118 |
| 10 | 1.00 | 1.00 | 1.00 | 119 |
| 11 | 1.00 | 1.00 | 1.00 | 118 |

**A graph of a logistic confusion matrix

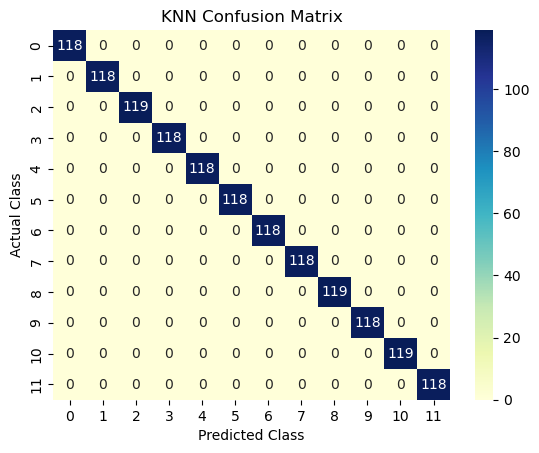
AI-generated content may be incorrect.**

**Overall Performance**

| **Metric** | **Score** | **Samples** |
| --- | --- | --- |
| Accuracy | 1.00 | 1419 |
| Macro Average | 1.00 | 1419 |
| Weighted Avg | 1.00 | 1419 |

**K – Nearest Neighbour**

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 1.00 | 1.00 | 1.00 | 118 |
| 1 | 1.00 | 1.00 | 1.00 | 118 |
| 2 | 1.00 | 1.00 | 1.00 | 119 |
| 3 | 1.00 | 1.00 | 1.00 | 118 |
| 4 | 1.00 | 1.00 | 1.00 | 118 |
| 5 | 1.00 | 1.00 | 1.00 | 118 |
| 6 | 1.00 | 1.00 | 1.00 | 118 |
| 7 | 1.00 | 1.00 | 1.00 | 118 |
| 8 | 1.00 | 1.00 | 1.00 | 119 |
| 9 | 1.00 | 1.00 | 1.00 | 118 |
| 10 | 1.00 | 1.00 | 1.00 | 119 |
| 11 | 1.00 | 1.00 | 1.00 | 118 |



| **Metric** | **Score** | **Samples** |
| --- | --- | --- |
| Accuracy | 1.00 | 1419 |
| Macro Average | 1.00 | 1419 |
| Weighted Avg | 1.00 | 1419 |

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 0.50 | 1.00 | 0.66 | 118 |
| 1 | 1.00 | 1.00 | 1.00 | 118 |
| 2 | 0.00 | 0.00 | 0.00 | 119 |
| 3 | 1.00 | 1.00 | 1.00 | 118 |
| 4 | 1.00 | 1.00 | 1.00 | 118 |
| 5 | 0.50 | 1.00 | 0.66 | 118 |
| 6 | 1.00 | 1.00 | 1.00 | 118 |
| 7 | 1.00 | 1.00 | 1.00 | 118 |
| 8 | 0.00 | 0.00 | 0.00 | 119 |
| 9 | 1.00 | 1.00 | 1.00 | 118 |
| 10 | 0.00 | 0.00 | 0.00 | 119 |
| 11 | 0.50 | 1.00 | 0.66 | 118 |

**Support Vector Machine**

A graph of a number of numbers

AI-generated content may be incorrect.

| **Metric** | **Value** | **Samples** |
| --- | --- | --- |
| Accuracy | 0.75 | 1419 |
| Macro Avg | 0.62 | 1419 |
| Weighted Avg | 0.62 | 1419 |

# Observations

For the given dataset:

* **Accuracy** order of the ML algorithms is observed to be: **Logistic = KNN > SVM**
* Increasing the **number of iterations** for algorithms for Logistic Regression and SVM helped in improving the accuracy of the models.
* Choosing the right kernel (here RBF/Gaussian is used) in SVM and right solver (here ‘lbfgs’ is used) in Logistic Regression algorithm is also an important parameter to get a good accuracy.
* While feature extraction, training and evaluation; it was observed that when the models were trained on RMS, MS, MAV and WL features, the accuracy was observed to be quite low (8-11%) but when the feature SSC was also used along with other features then the accuracy improved massively varying between 75-100% for different ML algorithms.
* Therefore, **SSC (Slope Sign Changes)** turned out to be the **most important feature** for training on EMG signals which contains the most important information for differentiating between gestures.

# Conclusion

The *PulseSort* project successfully demonstrates that EMG-based gesture recognition is viable and highly accurate when paired with effective preprocessing and feature engineering. Among all extracted features, Slope Sign Changes (SSC) emerged as the most decisive, significantly boosting performance across models. Logistic Regression and KNN achieved 100% classification accuracy across 12 gestures, confirming their suitability for this type of signal data. SVM’s comparatively lower performance highlights the importance of kernel selection and parameter tuning. This study emphasizes that model accuracy hinges not just on algorithm choice but also on thoughtful data segmentation and feature design. These findings can contribute to enhanced real-time gesture control systems for prosthetics, robotics, and human-computer interfaces.