ENGINNERING FACULTY - COMPUTER ENGINEERING DEPARTMENT

Biomedical Signal Analysis and Machine Learning
2023-2024

STUDENT NAME : Semih Utku POLAT
STUDENT ID : 191805060

TUDENT NAME : Burak Tüzel
STUDENT ID : 191805057

TABLE OF CONTENTS

1 Contents

INTRODUCTION

Electromyography (EMG) is a diagnostic technique used in medicine and biomedical engineering to assess and record the electrical activity produced by muscles during contraction and relaxation. It provides valuable insights into muscle function, helping to understand neuromuscular disorders, monitor rehabilitation progress, and analyze biomechanical aspects of movement.

2 Dataset Description

The dataset comprises raw electromyographic (EMG) data recorded using a MYO Thalmic bracelet worn on the forearm of 36 subjects. The bracelet is equipped with eight sensors that acquire myographic signals simultaneously and transmit them via Bluetooth to a PC. The dataset captures patterns while subjects performed static hand gestures. Each subject executed two series, each containing six or seven basic gestures, with each gesture lasting for 3 seconds and a 3-second pause between gestures.

Description of the dataset:

- Column 1: Time(time in ms)
- Column 2 to 9: Channel(eightEMG channels of MYO Thalmic bracelet)
- Column 10: Class(thelabel of gestures):
    - 0 - unmarked data,
    - 1 - hand at rest,
    - 2 - hand clenched in a fist,
    - 3 - wrist flexion,
    - 4 – wrist extension,
    - 5 – radial deviations,
    - 6 - ulnar deviations,
    - 7 - extended palm (the gesture was not performed by all subjects).

3 Dataset Preprocess

First things first, import the necessary libraries.

```
# Libraries
import matplotlib.pylab as plt
import os
import pandas as pd
import numpy as np
import time

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import pickle
from sklearn.inspection import permutation_importance
```

Read all the raw data rows from .txt files in the dataset and set them to all_data variable.

```
# Get data from dataset
folders = [folder.name for folder in os.scandir(dataset_path) if folder.is_dir()]

all_data = pd.DataFrame()

for folder in folders:
    folder_path = os.path.join(dataset_path, folder)
    files = [file.name for file in os.scandir(folder_path) if file.is_file()]
    print(folder, files)
    for file in files:
        file_path = os.path.join(folder_path, file)
        current_data = pd.read_csv(file_path, sep = '\t')
        all_data = pd.concat([all_data, current_data])

all_data = all_data.dropna()
```

Counts of the rows in dataset (in all_data variable).

Total rows in the dataset	4.237.907
Rows with class=0 (unmarked data)	2.725.157
Rows with class=7 (extended palm)	13.696

4 Feature Extraction

Define a function named extractFeatures() to **extract all time domain features from columns representing channel 1 to channel 8 in the dataset.**

Mean	Clearance Factor	Threshold Entropy	Mean Absolute Value Slope
Variance	Delta RMS	Sure Entropy	Mean of Amplitude
Standar Desviation	Root sum of Squares	Norm Entropy	Log RMS
Root Mean Square	Energy	Peak to peak	Conduction Velocity of Signal
Max Value	Latitude Factor	Minimum value	Average Amplitude Change
Kurtosis	Weighted SSR absolute	Peak value	V-ORDER 2
Skewness	Pulse Index	6th Statistical moment	V-ORDER 3
Kurtosis Energy Operator	Mean Square Error	Crest Factor	Maximum Fractal Length T
Absolute Media	Normalized Normal Negative Likelihoog	Integrated signal	Difference Absolute Strandard Deviation
CPT1	Mean Deviation	Square root amplitude value	Myopulse percentage rate
CPT2	Standard Deviation Impulse Factor	Simple Square Integral	Higher order Temporal Moments
CPT3	Log - Log Ratio	Zero crossing	Difference Absolute Variance Value
CPT4	Kth Central Moment	Wavelength	Margin Index
CPT5	Histogram lower bound	Wilson Amplitude	Waveform Indicators
CPT6	Histogram upper bound	Slope Sign Change	Weibull Negative Log-Likelihood
Fifth statistic moment	Normalized Moment	Log Detector	Pulse Indicators
Shape Factor	Shannon Entropy	Modified Mean Absolute Value 1	
Impulse Factor	Log energy entropy	Modified mean Absolute Value 2	

Some features trigger warnings in the dataset due to divisions by zero or log of zero, etc. Therefore, they have been commented out.

```python
# Feature operations
def extractFeatures(x):
    N = len(x)

    mean = np.mean(x)
    variance = np.var(x)
    std_deviation = np.std(x)
    root_mean_square = np.sqrt(np.mean(x**2))
    max_value = np.max(x)
    kurtosis = np.sum((x - mean)**4) / (N * std_deviation**4)
    skewness = np.sum((x - mean)**3) / (N * std_deviation**3)
    kurtosis_energy_operator = np.sum(np.diff(x)**4) / (N**2 * np.mean(np.diff(x)**2)**2)
    absolute_media = np.mean(np.abs(x))
    cpt1 = np.max(np.abs(x)) / np.sqrt(root_mean_square)
    cpt2 = np.max(np.abs(x)) / np.sqrt(np.mean(x**2))
    cpt3 = np.max(np.abs(x)) / np.mean(np.abs(x))
    cpt4 = np.sum(np.log(np.abs(x) + 1)) / (N * np.log(std_deviation + 1))
    cpt5 = np.sum(np.exp(x)) / (N * np.exp(std_deviation))
    cpt6 = np.sum(np.sqrt(np.abs(x))) / (N * np.sqrt(variance))
    fifth_statistic_moment = np.sum((x - mean)**5)
    shape_factor = root_mean_square / np.mean(np.abs(x))
    impulse_factor = np.max(np.abs(x)) / (1 / N * np.sum(np.abs(x)))
    clearance_factor = np.max(np.abs(x)) / (1 / N * np.sum(x**2))
    delta_rms = np.sqrt(np.mean(np.diff(x)**2))
    root_sum_of_squares = np.sqrt(np.sum(x**2))
    energy = np.sum(x**2)
    latitude_factor = np.max(np.abs(x)) / (1 / N * np.sum(np.sqrt(np.abs(x))))
    weighted_ssr_absolute = 1 / N * np.sum(np.sqrt(np.abs(x)))**2
    # pulse_index = np.max(x) / np.mean(x)
    mean_square_error = np.mean((x - mean)**2)
    normalized_normal_negative_likelihood = np.log(std_deviation / root_mean_square)
    mean_deviation = np.sum(np.abs(x - mean)) / (N * np.sqrt(variance))
    std_deviation_impulse_factor = std_deviation / np.mean(np.abs(x))
    log_log_ratio = 1 / np.log(std_deviation) * np.sum(np.log(np.abs(x) + 1))
    kth_central_moment = np.mean((x - np.mean(x))**3) # K is set to 3
    histogram_lower_bound = np.min(x) - 0.5 * (np.max(x) - np.min(x)) / (N - 1)
    histogram_upper_bound = np.max(x) + 0.5 * (np.max(x) - np.min(x)) / (N - 1)
    normalized_moment = np.sum((x - np.mean(x))**5) / np.sqrt(np.sum((x - np.mean(x))**2)**5)
    # shannon_entropy = -np.sum(np.log(x**2))
    # log_energy_entropy = np.sum(np.log(x**2))
    # threshold_entropy = np.where(np.abs(x) > 0.2, 1, 0)
    sure_entropy = len(x) - np.count_nonzero(np.abs(x) <= 0.2) + np.sum(np.minimum(x**2, 0.2**2))
    norm_entropy = np.sum(np.abs(x)**0.2)
    peak_to_peak = np.max(x) - np.min(x)
    minimum_value = np.min(x)
    peak_value = 0.5 * (np.max(x) - np.min(x))
    sixth_statistical_moment = np.sum((x - mean)**6)
    crest_factor = np.max(np.abs(x)) / root_mean_square
    integrated_signal = np.sum(np.abs(x))
    square_root_amplitude_value = (np.sum(np.sqrt(np.abs(x))) / N)**2
    simple_square_integral = np.sum(np.abs(x)**2)
    zero_crossing = np.sum(np.sign(-x[:-1] * x[1:]))
    wavelength = np.sum(np.abs(np.diff(x)))
    wilson_amplitude = np.sum(np.where(np.abs(np.diff(x)) - 0.2 > 0, 1, 0))
    slope_sign_change = np.sum(np.where((x[1:] - x[:-1]) * (x[1:] - np.roll(x, 1)[1:]) >= 0, 1, 0))
    # log_detector = np.exp(np.mean(np.log(np.abs(x))))
    modified_mean_absolute_value_1 = np.mean(np.where((0.25 * N <= np.arange(N)) & (np.arange(N) <= 0.75 * N), 1, 0) * np.abs(x))
    modified_mean_absolute_value_2 = np.mean(np.where((0.25 * N <= np.arange(N)) & (np.arange(N) <= 0.75 * N), 1, 0) * np.abs(x))
    mean_absolute_value_slope = np.mean(np.diff(x))
    mean_of_amplitude = np.sum(np.abs(np.diff(x)))
    log_rms = np.log(np.sqrt(np.sum(x**2)))
    conduction_velocity_signal = 1 / (N - 1) * np.sum(x**2)
    average_amplitude_change = 1 / N * np.sum(np.diff(x)**2)
    v_order_2 = np.sqrt(1 / N * np.sum(x**2))
    v_order_3 = np.cbrt(1 / N * np.sum(x**3))
    maximum_fractal_length = np.log10(np.sum(np.abs(np.diff(x))))
    difference_absolute_standard_deviation = np.sqrt(1 / (N - 1) * np.sum(np.diff(x)**2))
    myopulse_percentage_rate = np.mean(np.where(x >= 0.2, 1, 0))
    higher_order_temporal_moments = np.mean(np.abs(x)**3)
    difference_absolute_variance_value = 1 / (N - 2) * np.sum(np.diff(x)**2)
    # margin_index = (np.max(x) / np.sqrt(1 / N * np.sum(x)))**2
    waveform_indicators = np.sum(x) / N
    # weibull_negative_log_likelihood = -np.sum(np.log((0.2 * 1)**5 - np.sign(x) * x))
    pulse_indicators = np.max(x) / (1 / N * np.sum(np.abs(x)))


    return mean, variance, std_deviation, root_mean_square, max_value, kurtosis, skewness, kurtosis_energy_operator, absolute_media
```

Also, extract the mean of channel columns as features based on the window size.

Channel 1 mean	Channel 4 mean	Channel 7 mean
Channel 2 mean	Channel 5 mean	Channel 8 mean
Channel 3 mean	Channel 6 mean	

Define a function named getFeatures(all_data, window_size, window_hop, window_features_path). This function extracts features based on the specified window size and window stride parameters and saves the features as a CSV file.

The function also calculates label, percent and label2. These variables provide insights into the dominant class within a given window and its percentage distribution, aiding in the analysis of temporal patterns in the dataset.

- label: Represents the most frequent class within the given window. It is determined by calculating the bin count of class labels in the window and selecting the class with the highest count as the most frequent class.
- percent: Represents the percentage of the most frequent class within the window. It is calculated by dividing the count of the most frequent class by the total number of samples in the window.
- label2: Represents the second most frequent class within the window. If the most frequent class constitutes 100% of the window, then the second most frequent class is set to be the same as the most frequent class. Otherwise, the count of the most frequent class is set to zero, and the second most frequent class is determined based on the updated counts.

```python
def getFeatures(all_data, window_size, window_hop, window_features_path):

    features_list = []

    for i in range(0, len(all_data), window_hop):

        selmat = all_data.iloc[i:i+window_size, 1:-1].to_numpy().flatten()
        mean, variance, std_deviation, root_mean_square, max_value, kurtosis, skewness, kurtosis_energy_operator, absolute_media, cpt


        ch1mean = all_data.iloc[i:i+window_size,1].mean()
        ch2mean = all_data.iloc[i:i+window_size,2].mean()
        ch3mean = all_data.iloc[i:i+window_size,3].mean()
        ch4mean = all_data.iloc[i:i+window_size,4].mean()
        ch5mean = all_data.iloc[i:i+window_size,5].mean()
        ch6mean = all_data.iloc[i:i+window_size,6].mean()
        ch7mean = all_data.iloc[i:i+window_size,7].mean()
        ch8mean = all_data.iloc[i:i+window_size,8].mean()


        # flabel: the most frequent class
        bincountlist = np.bincount(all_data.iloc[i:i+window_size, -1].to_numpy(dtype='int64'))
        most_frequent_class = bincountlist.argmax()
        label = most_frequent_class

        # fpercent: the percentage of the most frequent class
        percentage_most_frequent=bincountlist[most_frequent_class] / len(all_data.iloc[i:i+window_size, -1].to_numpy(dtype='int64'))
        percent = percentage_most_frequent

        # flabel2: the second most frequent class
        if percentage_most_frequent == 1.0:
            most_frequent_class2 = most_frequent_class
        else:
            bincountlist[most_frequent_class] = 0
            most_frequent_class2=bincountlist.argmax()

        label2 = most_frequent_class2


        features_list.append({
            'mean': mean,
            'variance': variance,
            'std': std_deviation,
            'root_mean_square': root_mean_square,
            'max': max_value,
            'kurtosis': kurtosis
```

```python
            'wilson_amplitude': wilson_amplitude,
            'slope_sign_change': slope_sign_change,
            # 'log_detector': log_detector,
            'modified_mean_absolute_value_1': modified_mean_absolute_value_1,
            'modified_mean_absolute_value_2': modified_mean_absolute_value_2,
            'mean_absolute_value_slope': mean_absolute_value_slope,
            'mean_of_amplitude': mean_of_amplitude,
            'log_rms': log_rms,
            'conduction_velocity_signal': conduction_velocity_signal,
            'average_amplitude_change': average_amplitude_change,
            'v_order_2': v_order_2,
            'v_order_3': v_order_3,
            'maximum_fractal_length': maximum_fractal_length,
            'difference_absolute_standard_deviation': difference_absolute_standard_deviation,
            'myopulse_percentage_rate': myopulse_percentage_rate,
            'higher_order_temporal_moments': higher_order_temporal_moments,
            'difference_absolute_variance_value': difference_absolute_variance_value,
            # 'margin_index': margin_index,
            'waveform_indicators': waveform_indicators,
            # 'weibull_negative_log_likelihood': weibull_negative_log_likelihood,
            'pulse_indicators': pulse_indicators,

            'ch1mean': ch1mean,
            'ch2mean': ch2mean,
            'ch3mean': ch3mean,
            'ch4mean': ch4mean,
            'ch5mean': ch5mean,
            'ch6mean': ch6mean,
            'ch7mean': ch7mean,
            'ch8mean': ch8mean,

            'label': label,
            'percent': percent,
            '2ndlabel': label2
        })


    # Save the features
    rdf = pd.DataFrame(features_list)
    rdf.to_csv(f'{window_features_path}/emg_gesture_ws{window_size}_hop{window_hop}.csv', index = None, header = True)
    print(f'Created: emg_gesture_ws{window_size}_hop{window_hop}.csv')
    # rdf.info()

    return rdf
```

mean	variance	std	root_mear	max	kurtosis	skewness	kurtosis_e	absolute_r	cpt1	cpt2	cpt3	cpt4	cpt5	cpt6	fifth_statis	shape_fac	impulse_fa	clearance_	delta_rms	root_sum_	energy	latitude_fa
-9.48E-06	3.19E-10	1.79E-05	2.02E-05	4.00E-05	5.286698	-0.60785	0.000478	1.47E-05	0.022238	4.945286	6.78196	0.825397	0.999973	181.3696	-1.42E-19	1.371399	6.78196	244558.6	1.92E-05	0.001809	3.27E-06	0.030864
-9.53E-06	3.02E-10	1.74E-05	1.98E-05	4.00E-05	5.430979	-0.64446	0.000474	1.45E-05	0.022458	5.043611	6.917423	0.831557	0.999973	184.3094	-1.36E-19	1.371522	6.917423	254380.1	1.87E-05	0.001773	3.14E-06	0.03121
-8.81E-06	2.62E-10	1.62E-05	1.84E-05	6.00E-05	4.727304	-0.32235	0.00044	1.38E-05	0.023294	5.426255	7.267442	0.850006	0.999975	194.0915	-4.93E-20	1.339311	7.267442	294442.4	1.79E-05	0.001648	2.72E-06	0.031827
-8.69E-06	2.44E-10	1.56E-05	1.79E-05	6.00E-05	4.362126	-0.21341	0.000448	1.35E-05	0.016553	3.914113	5.195287	0.861883	0.999976	198.7773	-2.43E-20	1.327322	5.195287	218861.1	1.74E-05	0.0016	2.56E-06	0.022526
-8.62E-06	2.45E-10	1.57E-05	1.79E-05	6.00E-05	4.415521	-0.20921	0.000457	1.35E-05	0.016556	3.91572	5.180389	0.862726	0.999976	199.4617	-2.59E-20	1.322972	5.180389	219040.9	1.77E-05	0.001599	2.56E-06	0.022407
-8.93E-06	2.48E-10	1.57E-05	1.81E-05	6.00E-05	4.338532	-0.21036	0.000462	1.36E-05	0.016452	3.866798	5.133376	0.866073	0.999975	198.4496	-2.38E-20	1.327552	5.133376	213601.9	1.77E-05	0.001619	2.62E-06	0.022403
-8.86E-06	2.48E-10	1.58E-05	1.81E-05	6.00E-05	4.358893	-0.2365	0.000457	1.36E-05	0.016462	3.871231	5.135259	0.864736	0.999975	198.6127	-2.65E-20	1.326518	5.135259	214091.8	1.78E-05	0.001617	2.62E-06	0.022358
-9.14E-06	2.31E-10	1.52E-05	1.77E-05	6.00E-05	4.290176	-0.42655	0.000462	1.34E-05	0.016622	3.947187	5.240502	0.878844	0.999976	203.8767	-3.83E-20	1.327655	5.240502	222575.5	1.76E-05	0.001586	2.52E-06	0.02259
-9.18E-06	2.29E-10	1.51E-05	1.77E-05	6.00E-05	4.334111	-0.44642	0.000474	1.33E-05	0.016642	3.956397	5.261674	0.879717	0.999976	204.4387	-3.87E-20	1.329916	5.261674	223615.4	1.73E-05	0.001582	2.50E-06	0.022642
-8.89E-06	2.34E-10	1.53E-05	1.77E-05	6.00E-05	4.333282	-0.3877	0.000467	1.33E-05	0.016637	3.954108	5.270092	0.867701	0.999976	200.9699	-3.61E-20	1.332814	5.270092	223356.7	1.72E-05	0.001583	2.51E-06	0.022754
-8.37E-06	2.24E-10	1.50E-05	1.71E-05	6.00E-05	4.186972	-0.32508	0.000455	1.29E-05	0.016908	4.08422	5.440062	0.860413	0.999977	201.3473	-2.73E-20	1.331971	5.440062	238297.9	1.71E-05	0.001533	2.35E-06	0.023247
-8.16E-06	2.22E-10	1.49E-05	1.70E-05	6.00E-05	4.237069	-0.34807	0.000462	1.27E-05	0.01698	4.118983	5.501523	0.853484	0.999977	200.4976	-2.88E-20	1.335651	5.501523	242371.8	1.70E-05	0.00152	2.31E-06	0.023419
-7.92E-06	2.09E-10	1.45E-05	1.65E-05	6.00E-05	4.285828	-0.29317	0.000475	1.24E-05	0.017236	4.243788	5.639476	0.85771	0.999978	204.2327	-2.34E-20	1.328878	5.639476	257282	1.65E-05	0.001475	2.18E-06	0.023684
-8.34E-06	1.99E-10	1.41E-05	1.64E-05	6.00E-05	4.236732	-0.36338	0.00048	1.24E-05	0.017291	4.271351	5.654281	0.877365	0.999978	209.6527	-2.53E-20	1.323769	5.654281	260634.8	1.62E-05	0.001466	2.15E-06	0.023662
-8.34E-06	1.90E-10	1.38E-05	1.61E-05	6.00E-05	4.385065	-0.40293	0.000431	1.22E-05	0.017446	4.348225	5.753622	0.88369	0.999978	213.1886	-2.57E-20	1.323212	5.753622	270100.8	1.59E-05	0.00144	2.07E-06	0.023849

latitude_fa	weighted_	mean_squ	normalizec	mean_dev	std_deviat	log_log_ra	kth_centra	histogram_	histogram_	normalizec	sure_entro	norm_entr	peak_to_p	minimum_	peak_valu	sixth_stati	crest_fact	integrated_	square_ro	simple_squ	zero_cross	wavelengt
0.030864	0.083981	3.19E-10	-0.12394	0.708979	1.211533	-0.01079	-3.47E-15	-0.0001	4.00E-05	-1.37E-05	3.27E-06	687.753	0.00014	-0.0001	7.00E-05	1.38E-23	4.945286	0.11796	1.05E-05	3.27E-06	-2525	0.11658
0.03121	0.082131	3.02E-10	-0.13147	0.708444	1.202558	-0.01055	-3.39E-15	-0.0001	4.00E-05	-1.50E-05	3.14E-06	683.0769	0.00014	-0.0001	7.00E-05	1.27E-23	5.043611	0.11565	1.03E-05	3.14E-06	-2567	0.11312
0.031827	0.078975	2.62E-10	-0.12965	0.732626	1.176457	-0.00998	-1.37E-15	-0.0001	6.00E-05	-7.75E-06	2.72E-06	678.8167	0.00016	-0.0001	8.00E-05	6.47E-24	5.426255	0.11008	9.87E-06	2.72E-06	-2465	0.1097
0.022526	0.07725	2.44E-10	-0.13453	0.744924	1.160246	-0.00974	-8.15E-16	-7.00E-05	6.00E-05	-4.54E-06	2.56E-06	673.7337	0.00013	-7.00E-05	6.50E-05	4.20E-24	3.914113	0.10779	9.66E-06	2.56E-06	-2487	0.10604
0.022407	0.078078	2.45E-10	-0.13223	0.741961	1.159111	-0.00977	-8.04E-16	-7.00E-05	6.00E-05	-4.80E-06	2.56E-06	678.236	0.00013	-7.00E-05	6.50E-05	4.32E-24	3.91572	0.1081	9.76E-06	2.56E-06	-2437	0.10711
0.022403	0.078103	2.48E-10	-0.13955	0.739119	1.154633	-0.00986	-8.21E-16	-7.00E-05	6.00E-05	-4.30E-06	2.62E-06	674.9041	0.00013	-7.00E-05	6.50E-05	4.23E-24	3.866798	0.10909	9.76E-06	2.62E-06	-2537	0.1069
0.022358	0.078416	2.48E-10	-0.13723	0.742509	1.156417	-0.00986	-9.26E-16	-7.00E-05	6.00E-05	-4.76E-06	2.62E-06	677.2936	0.00013	-7.00E-05	6.50E-05	4.32E-24	3.871231	0.10905	9.80E-06	2.62E-06	-2510	0.10768
0.02259	0.076815	2.31E-10	-0.15427	0.741625	1.137854	-0.00963	-1.50E-15	-7.00E-05	6.00E-05	-8.24E-06	2.52E-06	673.9926	0.00013	-7.00E-05	6.50E-05	3.55E-24	3.947187	0.10686	9.60E-06	2.52E-06	-2417	0.10649
0.022642	0.076467	2.29E-10	-0.15696	0.737009	1.136725	-0.00959	-1.54E-15	-7.00E-05	6.00E-05	-8.54E-06	2.50E-06	673.4088	0.00013	-7.00E-05	6.50E-05	3.52E-24	3.956397	0.10643	9.56E-06	2.50E-06	-2512	0.1047
0.022754	0.075713	2.34E-10	-0.14539	0.747309	1.152466	-0.00958	-1.39E-15	-7.00E-05	6.00E-05	-7.51E-06	2.51E-06	668.4485	0.00013	-7.00E-05	6.50E-05	3.76E-24	3.954108	0.10626	9.46E-06	2.51E-06	-2482	0.10416
0.023247	0.072536	2.24E-10	-0.13632	0.763283	1.162228	-0.00927	-1.09E-15	-7.00E-05	6.00E-05	-6.38E-06	2.35E-06	657.1848	0.00013	-7.00E-05	6.50E-05	3.12E-24	4.08422	0.10294	9.07E-06	2.35E-06	-2249	0.10351
0.023419	0.071473	2.22E-10	-0.13099	0.766197	1.171664	-0.00916	-1.15E-15	-7.00E-05	6.00E-05	-6.84E-06	2.31E-06	653.8448	0.00013	-7.00E-05	6.50E-05	3.15E-24	4.118983	0.10179	8.93E-06	2.31E-06	-2135	0.10272
0.023684	0.069884	2.09E-10	-0.13085	0.770614	1.165891	-0.00891	-8.89E-16	-7.00E-05	6.00E-05	-6.45E-06	2.18E-06	650.8205	0.00013	-7.00E-05	6.50E-05	2.81E-24	4.243788	0.0993	8.74E-06	2.18E-06	-2197	0.09943
0.023662	0.070011	1.99E-10	-0.14965	0.769418	1.139772	-0.00887	-1.02E-15	-7.00E-05	6.00E-05	-7.90E-06	2.15E-06	652.3887	0.00013	-7.00E-05	6.50E-05	2.46E-24	4.271351	0.09904	8.75E-06	2.15E-06	-2276	0.09754
0.023849	0.068917	1.90E-10	-0.15642	0.766877	1.131615	-0.0087	-1.05E-15	-7.00E-05	6.00E-05	-9.09E-06	2.07E-06	650.7014	0.00013	-7.00E-05	6.50E-05	2.35E-24	4.348225	0.09733	8.61E-06	2.07E-06	-2255	0.09661

wilson_am	slope_sign	modified_	modified_	mean_abs	mean_of_	log_rms	conductio	average_a	v_order_2	v_order_3	maximum_	difference	myopulse_	higher_orc	difference	waveform	pulse_indi	ch1mean	ch2mean	ch3mean	ch4mean	ch5mean
0	7999	7.30E-06	7.30E-06	-2.50E-09	0.11658	-6.31518	4.09E-10	3.70E-10	2.02E-05	-2.37E-05	-0.93338	1.92E-05	0	1.63E-14	3.70E-10	-9.48E-06	2.712784	-8.74E-06	-9.41E-06	-1.09E-05	-1.05E-05	-9.73E-06
0	7999	7.44E-06	7.44E-06	1.87E-22	0.11312	-6.33486	3.93E-10	3.50E-10	1.98E-05	-2.35E-05	-0.94646	1.87E-05	0	1.54E-14	3.50E-10	-9.53E-06	2.766969	-8.36E-06	-1.01E-05	-1.09E-05	-1.06E-05	-9.41E-06
0	7999	7.45E-06	7.45E-06	6.18E-23	0.1097	-6.40799	3.40E-10	3.21E-10	1.84E-05	-2.08E-05	-0.95979	1.79E-05	0	1.15E-14	3.21E-10	-8.81E-06	4.360465	-7.97E-06	-9.02E-06	-9.56E-06	-8.55E-06	-8.80E-06
0	7999	7.31E-06	7.31E-06	-2.50E-09	0.10604	-6.43801	3.20E-10	3.04E-10	1.79E-05	-1.99E-05	-0.97453	1.74E-05	0	1.02E-14	3.04E-10	-8.69E-06	4.453103	-8.37E-06	-9.56E-06	-9.68E-06	-7.29E-06	-8.57E-06
0	7999	7.28E-06	7.28E-06	1.25E-09	0.10711	-6.43842	3.20E-10	3.14E-10	1.79E-05	-1.98E-05	-0.97017	1.77E-05	0	1.02E-14	3.14E-10	-8.62E-06	4.440333	-8.46E-06	-9.44E-06	-9.51E-06	-7.43E-06	-8.80E-06
0	7999	6.81E-06	6.81E-06	4.66E-23	0.1069	-6.42584	3.28E-10	3.12E-10	1.81E-05	-2.01E-05	-0.97102	1.77E-05	0	1.05E-14	3.12E-10	-8.93E-06	4.400037	-8.83E-06	-9.42E-06	-9.67E-06	-8.01E-06	-9.45E-06
0	7999	6.73E-06	6.73E-06	-5.00E-09	0.10768	-6.42699	3.27E-10	3.15E-10	1.81E-05	-2.02E-05	-0.96786	1.78E-05	0	1.05E-14	3.15E-10	-8.86E-06	4.401651	-8.63E-06	-8.90E-06	-9.22E-06	-7.94E-06	-8.97E-06
0	7999	6.40E-06	6.40E-06	1.25E-09	0.10649	-6.44642	3.15E-10	3.08E-10	1.77E-05	-2.05E-05	-0.97269	1.76E-05	0	1.00E-14	3.08E-10	-9.14E-06	4.491859	-8.69E-06	-9.32E-06	-1.03E-05	-8.61E-06	-9.00E-06
0	7999	6.33E-06	6.33E-06	-2.50E-09	0.1047	-6.44875	3.13E-10	3.00E-10	1.77E-05	-2.05E-05	-0.98005	1.73E-05	0	1.00E-14	3.00E-10	-9.18E-06	4.510007	-8.20E-06	-9.61E-06	-1.04E-05	-8.65E-06	-9.05E-06
0	7999	6.25E-06	6.25E-06	2.50E-09	0.10416	-6.44817	3.13E-10	2.97E-10	1.77E-05	-2.03E-05	-0.9823	1.72E-05	0	9.99E-15	2.97E-10	-8.89E-06	4.517222	-7.73E-06	-8.60E-06	-1.02E-05	-8.10E-06	-9.02E-06
0	7999	6.34E-06	6.34E-06	1.25E-09	0.10351	-6.48055	2.94E-10	2.94E-10	1.71E-05	-1.94E-05	-0.98502	1.71E-05	0	8.95E-15	2.94E-10	-8.37E-06	4.66291	-7.72E-06	-7.31E-06	-9.63E-06	-7.67E-06	-8.38E-06
0	7999	6.20E-06	6.20E-06	2.50E-09	0.10272	-6.48902	2.89E-10	2.90E-10	1.70E-05	-1.93E-05	-0.98834	1.70E-05	0	8.79E-15	2.90E-10	-8.16E-06	4.715591	-6.97E-06	-7.89E-06	-9.43E-06	-7.38E-06	-8.39E-06
0	7999	5.92E-06	5.92E-06	3.75E-09	0.09943	-6.51887	2.72E-10	2.73E-10	1.65E-05	-1.85E-05	-1.00248	1.65E-05	0	7.97E-15	2.73E-10	-7.92E-06	4.833837	-6.40E-06	-6.41E-06	-8.19E-06	-6.49E-06	-8.69E-06
0	7999	5.99E-06	5.99E-06	-5.00E-09	0.09754	-6.52535	2.69E-10	2.62E-10	1.64E-05	-1.87E-05	-1.01082	1.62E-05	0	7.78E-15	2.62E-10	-8.34E-06	4.846527	-7.43E-06	-7.43E-06	-8.52E-06	-7.06E-06	-8.73E-06
0	7999	6.00E-06	6.00E-06	1.25E-09	0.09661	-6.54318	2.59E-10	2.52E-10	1.61E-05	-1.85E-05	-1.01498	1.59E-05	0	7.43E-15	2.53E-10	-8.34E-06	4.931676	-7.71E-06	-7.14E-06	-9.66E-06	-6.78E-06	-8.59E-06

1	of_	log_rms	conductio	average_a	v_order_2	v_order_3	maximum_	difference	myopulse_	higher_orc	difference	waveform	pulse_indi	ch1mean	ch2mean	ch3mean	ch4mean	ch5mean	ch6mean	ch7mean	ch8mean	label	percent	2ndlabel
2	58	-6.31518	4.09E-10	3.70E-10	2.02E-05	-2.37E-05	-0.93338	1.92E-05	0	1.63E-14	3.70E-10	-9.48E-06	2.712784	-8.74E-06	-9.41E-06	-1.09E-05	-1.05E-05	-9.73E-06	-1.01E-05	-8.14E-06	-8.35E-06	0	1	0
3	12	-6.33486	3.93E-10	3.50E-10	1.98E-05	-2.35E-05	-0.94646	1.87E-05	0	1.54E-14	3.50E-10	-9.53E-06	2.766969	-8.36E-06	-1.01E-05	-1.09E-05	-1.06E-05	-9.41E-06	-1.02E-05	-8.09E-06	-8.62E-06	0	1	0
4	97	-6.40799	3.40E-10	3.21E-10	1.84E-05	-2.08E-05	-0.95979	1.79E-05	0	1.15E-14	3.21E-10	-8.81E-06	4.360465	-7.97E-06	-9.02E-06	-9.56E-06	-8.55E-06	-8.80E-06	-9.74E-06	-8.45E-06	-8.37E-06	0	1	0
5	04	-6.43801	3.20E-10	3.04E-10	1.79E-05	-1.99E-05	-0.97453	1.74E-05	0	1.02E-14	3.04E-10	-8.69E-06	4.453103	-8.37E-06	-9.56E-06	-9.68E-06	-7.29E-06	-8.57E-06	-9.45E-06	-8.39E-06	-8.18E-06	0	1	0
6	11	-6.43842	3.20E-10	3.14E-10	1.79E-05	-1.98E-05	-0.97017	1.77E-05	0	1.02E-14	3.14E-10	-8.62E-06	4.440333	-8.46E-06	-9.44E-06	-9.51E-06	-7.43E-06	-8.80E-06	-8.89E-06	-7.99E-06	-8.42E-06	0	1	0
7	69	-6.42584	3.28E-10	3.12E-10	1.81E-05	-2.01E-05	-0.97102	1.77E-05	0	1.05E-14	3.12E-10	-8.93E-06	4.400037	-8.83E-06	-9.42E-06	-9.67E-06	-8.01E-06	-9.45E-06	-9.16E-06	-8.01E-06	-8.92E-06	0	1	0
8	68	-6.42699	3.27E-10	3.15E-10	1.81E-05	-2.02E-05	-0.96786	1.78E-05	0	1.05E-14	3.15E-10	-8.86E-06	4.401651	-8.63E-06	-8.90E-06	-9.22E-06	-7.94E-06	-8.97E-06	-9.48E-06	-8.30E-06	-9.43E-06	0	1	0
9	49	-6.44642	3.15E-10	3.08E-10	1.77E-05	-2.05E-05	-0.97269	1.76E-05	0	1.00E-14	3.08E-10	-9.14E-06	4.491859	-8.69E-06	-9.32E-06	-1.03E-05	-8.61E-06	-9.00E-06	-8.98E-06	-8.37E-06	-9.84E-06	0	1	0
10	47	-6.44875	3.13E-10	3.00E-10	1.77E-05	-2.05E-05	-0.98005	1.73E-05	0	1.00E-14	3.00E-10	-9.18E-06	4.510007	-8.20E-06	-9.61E-06	-1.04E-05	-8.65E-06	-9.05E-06	-9.05E-06	-8.63E-06	-9.87E-06	0	1	0
11	16	-6.44817	3.13E-10	2.97E-10	1.77E-05	-2.03E-05	-0.9823	1.72E-05	0	9.99E-15	2.97E-10	-8.89E-06	4.517222	-7.73E-06	-8.60E-06	-1.02E-05	-8.10E-06	-9.02E-06	-8.95E-06	-8.62E-06	-9.93E-06	0	1	0
12	51	-6.48055	2.94E-10	2.94E-10	1.71E-05	-1.94E-05	-0.98502	1.71E-05	0	8.95E-15	2.94E-10	-8.37E-06	4.66291	-7.72E-06	-7.31E-06	-9.63E-06	-7.67E-06	-8.38E-06	-9.27E-06	-7.92E-06	-9.08E-06	0	1	0
13	72	-6.48902	2.89E-10	2.90E-10	1.70E-05	-1.93E-05	-0.98834	1.70E-05	0	8.79E-15	2.90E-10	-8.16E-06	4.715591	-6.97E-06	-7.89E-06	-9.43E-06	-7.38E-06	-8.39E-06	-9.49E-06	-6.91E-06	-8.81E-06	0	1	0
14	43	-6.51887	2.72E-10	2.73E-10	1.65E-05	-1.85E-05	-1.00248	1.65E-05	0	7.97E-15	2.73E-10	-7.92E-06	4.833837	-6.40E-06	-6.41E-06	-8.19E-06	-6.49E-06	-8.69E-06	-9.51E-06	-8.47E-06	-9.16E-06	0	1	0
15	54	-6.52535	2.69E-10	2.62E-10	1.64E-05	-1.87E-05	-1.01082	1.62E-05	0	7.78E-15	2.62E-10	-8.34E-06	4.846527	-7.43E-06	-7.43E-06	-8.52E-06	-7.06E-06	-8.73E-06	-9.39E-06	-8.61E-06	-9.51E-06	0	1	0
16	61	-6.54318	2.59E-10	2.52E-10	1.61E-05	-1.85E-05	-1.01498	1.59E-05	0	7.43E-15	2.53E-10	-8.34E-06	4.931676	-7.71E-06	-7.14E-06	-9.66E-06	-6.78E-06	-8.59E-06	-9.25E-06	-8.19E-06	-9.43E-06	0	1	0

5 Optimize Window Sizes

Attempt the process with varying window sizes while keeping the window stride constant. Compute the mean of cross-validation scores for each window size and record the time taken for each iteration. Determine the optimal window size based on the ratio of the mean cross-validation scores to the time taken.

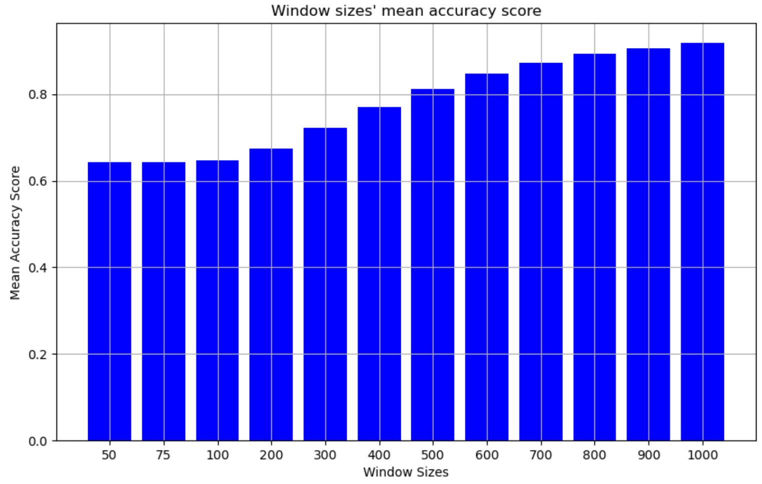
Window sizes below 50 result in mean of cross-validation scores below 0.64, while those above 1000 lead to overfitting with small size of data and labels. Therefore, these are excluded.

Additionally, keeping the window size and window stride equal did not yield satisfactory results. Therefore, the window stride remains constant.

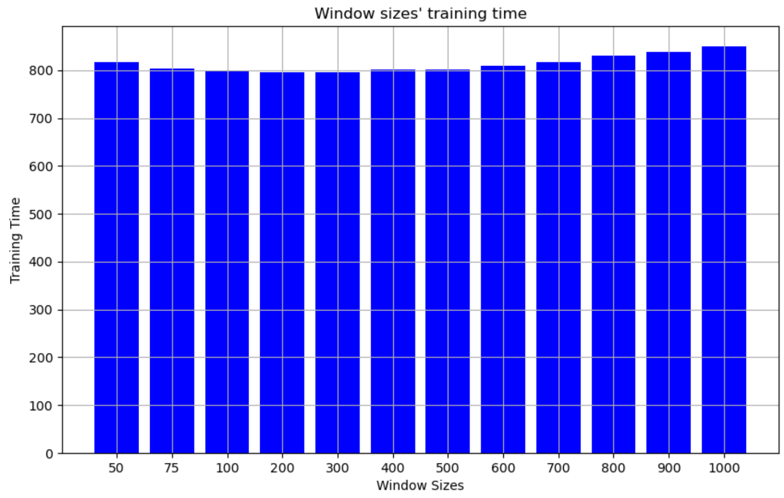
```
# Optimization of best window size
window_size_times = []
window_size_results = []
window_size_performance = []
window_sizes = [50, 75, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]

for window_size in window_sizes:
    start_time = time.time()
    window_hop = 50


    rdf = getFeatures(all_data, window_size, window_hop, window_size_features_path)

    X = rdf.drop(columns = ['label', 'percent', '2ndlabel']) # Features
    y = rdf['label'] # Class

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 50605057)

    rf_classifier = RandomForestClassifier()
    scores = cross_val_score(rf_classifier, X_train, y_train, cv = 10)
    mean_accuracy_score = np.mean(scores)


    total_time = time.time() - start_time
    window_size_times.append(total_time)
    window_size_results.append(mean_accuracy_score)
    window_size_performance.append(mean_accuracy_score / total_time)

    print('Window Size:', window_size, 'Window Hop:', window_hop, 'Mean Accuracy Score:', mean_accuracy_score)
    print('')

# Visualize the window size results
plotBar("Window sizes' mean accuracy score", window_size_results, window_sizes, 'Window Sizes', 'Mean Accuracy
plotBar("Window sizes' training time", window_size_times, window_sizes, 'Window Sizes', 'Training Time', f'{win
plotBar("Window sizes' mean accuracy score / training time", window_size_performance, window_sizes, 'Window Siz
```

Window Size - Window Hop	Mean Accuracy Score
50 - 50	0.6433543471792253
75 - 50	0.6434330786473357
100 - 50	0.6477196485439085
200 - 50	0.6743704111687642
300 - 50	0.7214976190333809
400 - 50	0.7695684858435262
500 - 50	0.8130775912133366
600 - 50	0.8479216641739702
700 – 50	0.8723437937222819
800 - 50	0.8927414387583383
900 - 50	0.9065584032941396
1000 - 50	**0.9186581825635255**

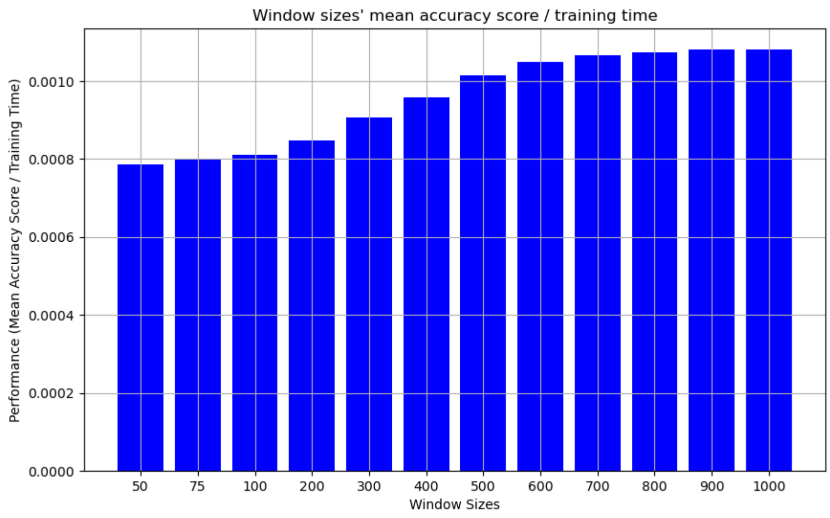
Mean accuracy scores for different window sizes: **Best window size selected as 1000**



Time taken:



Performance = Mean accuracy scores / time taken:



6 Optimize Window Strides For The Best Window Size

After finding the best window size.

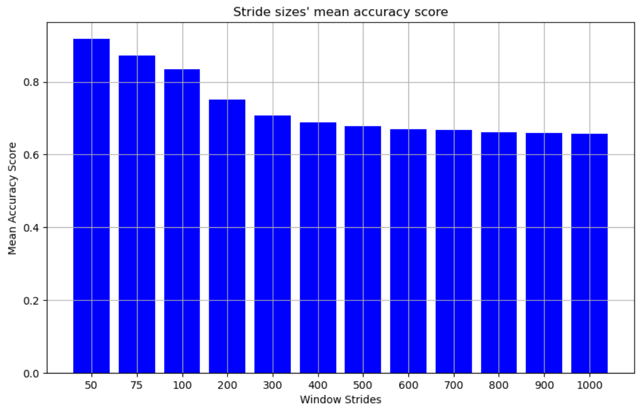
```
# Find the best window size
best_window_size = window_sizes[np.argmax(window_size_performance)]
print('Best window size:', best_window_size)
print('')
```

Attempt the same process with varying window strides while maintaining a constant window size determined as the best window size. And find the best window stride.

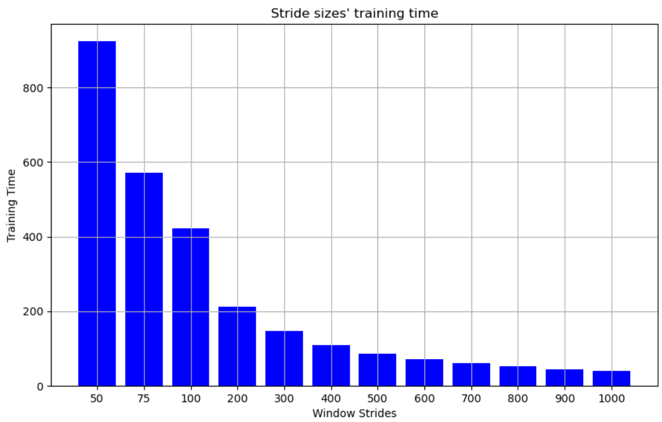
```
# Optimization of best hop/stride size for the best window size
window_hop_times = []
window_hop_results = []
window_hop_performance = []
window_hops = [25, 50, 75, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]

for window_hop in window_hops:
    start_time = time.time()


    rdf = getFeatures(all_data, best_window_size, window_hop, window_hop_features_path)

    X = rdf.drop(columns = ['label', 'percent', '2ndlabel']) # Features
    y = rdf['label'] # Class

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 50605057)

    rf_classifier = RandomForestClassifier()
    scores = cross_val_score(rf_classifier, X_train, y_train, cv = 10)
    mean_accuracy_score = np.mean(scores)


    total_time = time.time() - start_time
    window_hop_times.append(total_time)

    window_hop_results.append(mean_accuracy_score)
    window_hop_performance.append(mean_accuracy_score / total_time)

    print('Window Size:', best_window_size, 'Window Hop:', window_hop, 'Mean Accuracy Score:', mean_accuracy_score)
    print('')

# Visualize the window hop results
plotBar("Stride sizes' mean accuracy score", window_hop_results, window_hops, 'Window Strides', 'Mean Accuracy Score', f
plotBar("Stride sizes' training time", window_hop_times, window_hops, 'Window Strides', 'Training Time', f'{window_hop_f
plotBar("Stride sizes' mean accuracy score / training time", window_hop_performance, window_hops, 'Window Strides', 'Per

# Find the best window hop for the best window size
best_window_hop = window_hops[np.argmax(window_hop_performance)]
print('Best window size:', best_window_size, 'Best window hop:', best_window_hop)
print('')
```

Window Size - Window Hop	Mean Accuracy Score
1000 - 50	**0.9180158071933586**
1000 - 75	0.8714779422255784
1000 - 100	0.8335690678132094
1000 - 200	0.7502494396128575
1000 - 300	0.7076450845914424
1000 - 400	0.6879924589897071
1000 - 500	0.6786827105327431
1000 - 600	0.6701295003218937
1000 – 700	0.6683799919050188
1000 - 800	0.6621284991984215
1000 - 900	0.6595120210535705
1000 - 1000	0.6578444710118042

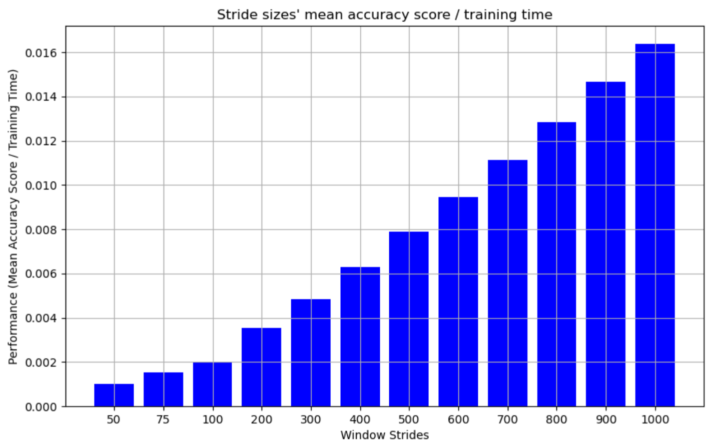
Mean accuracy scores for different window strides: **Best window stride selected as 50**



Time taken:



Performance = Mean accuracy scores / time taken:



7 Feature Selection (Feature Importances)

Load the dataframe containing features with the best window size and its optimal stride. Assign features to the 'X' variable and labels (class) to the 'y' variable. Split the data and fit the random forest classifier.

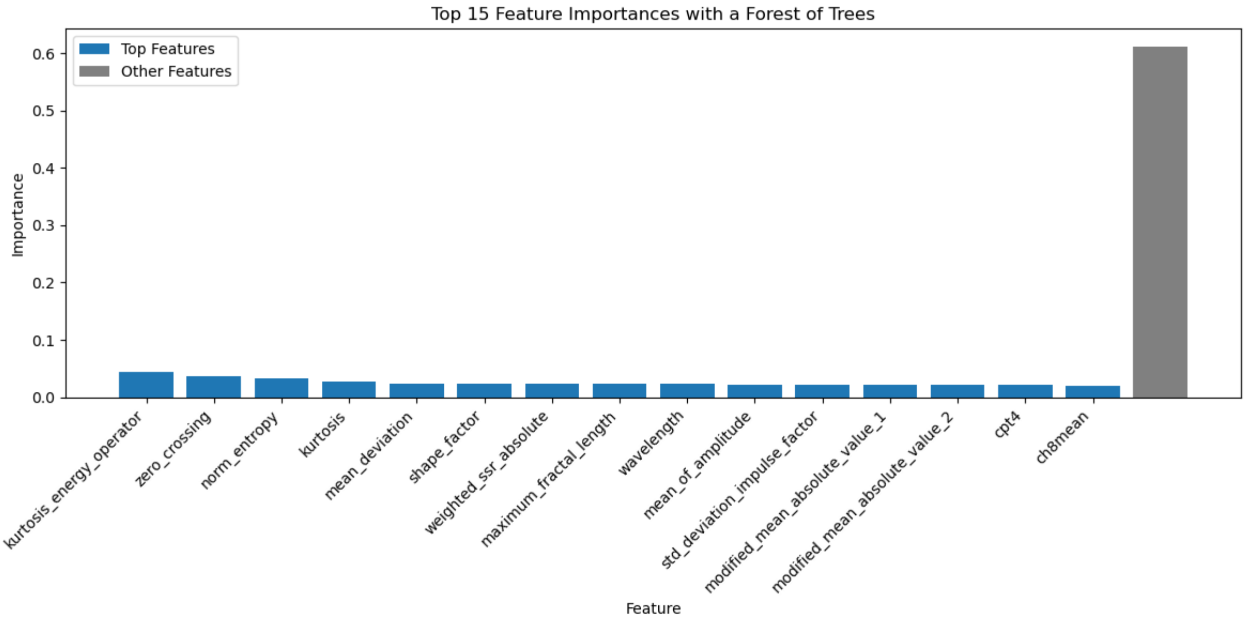
```
# Get the best window size and its hop
best_rdf = pd.read_csv(f'{window_hop_features_path}/emg_gesture_ws{best_window_size}_hop{best_window_hop}.csv')

X = best_rdf.drop(columns = ['label', 'percent', '2ndlabel']) # Features
y = best_rdf['label'] # Class

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 50605057)

rf_classifier = RandomForestClassifier(n_estimators = 100)
rf_classifier.fit(X_train, y_train)
```

1) First method of feature selection

```
# Get feature importances
importances = rf_classifier.feature_importances_
importances_df = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
importances_df = importances_df.sort_values(by = 'Importance', ascending = False)

print("Feature importances with a forest of trees (sorted by importance):")
for index, row in importances_df.iterrows():
    print(row['Feature'], row['Importance'])

plt.figure(figsize=(10, 6))
plt.bar(range(len(importances_df)), importances_df['Importance'], tick_label = importances_df['Feature'])
plt.title("Feature importances with a forest of trees (sorted by importance)")
plt.xticks(rotation = 45, ha = 'right')
plt.tight_layout()
plt.savefig(f'{main_path}/feature_importances.png')
plt.show()
```

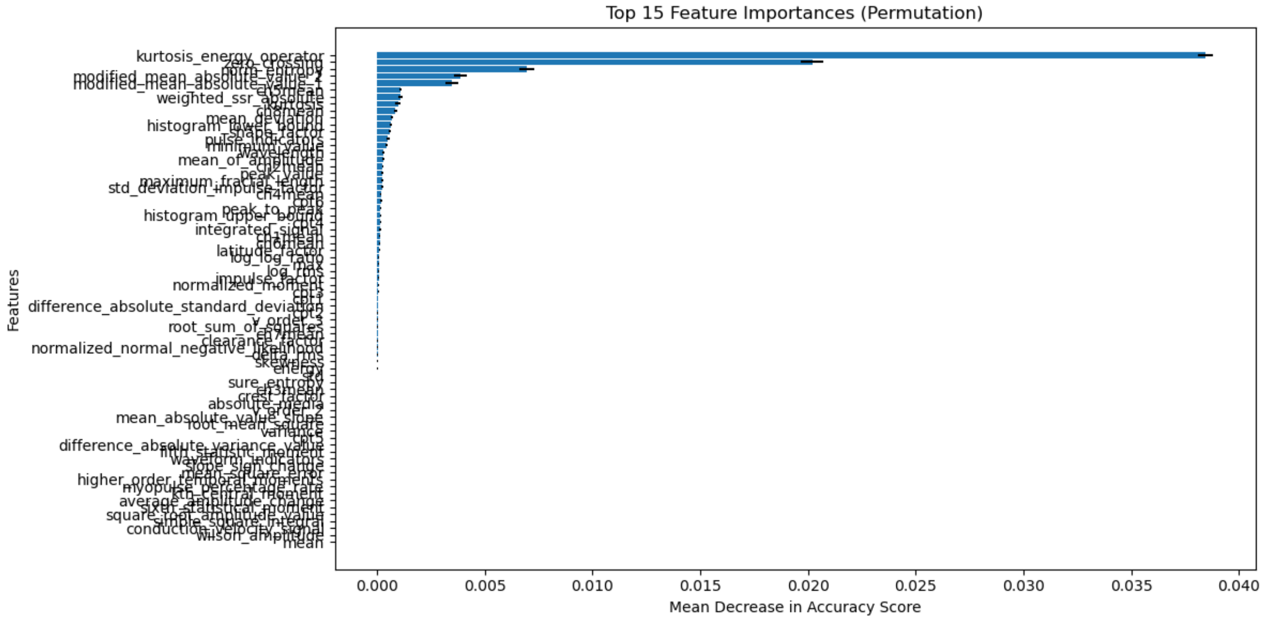
Feature importances with a forest of trees:

```
kurtosis_energy_operator 0.04354961752353065
zero_crossing 0.03656286521657384
norm_entropy 0.03244613003176572
kurtosis 0.027732992392819934
mean_deviation 0.024258786767832396
shape_factor 0.023705827256619908
weighted_ssr_absolute 0.023583193916474912
maximum_fractal_length 0.022801075703969144
wavelength 0.022584071981332995
mean_of_amplitude 0.022494531893615367
std_deviation_impulse_factor 0.0224056955903768
modified_mean_absolute_value_1 0.022009079446939417
modified_mean_absolute_value_2 0.02181102903949158
cpt4 0.021335515093409866
ch8mean 0.02048495804234275
ch5mean 0.020422137847354346
pulse_indicators 0.02026074284422262
cpt6 0.01976972211403803
ch4mean 0.01851182544865337
normalized_moment 0.01800456683249199
skewness 0.017762822768285527
latitude_factor 0.017245136672398013
integrated_signal 0.017211387142289324
ch2mean 0.017181724420180577
ch1mean 0.016727116014472063
delta_rms 0.016398284951723914
ch7mean 0.0160889285695412
histogram_lower_bound 0.01600988837174102
difference_absolute_standard_deviation 0.01583028088917476
cpt3 0.015735789316094748
minimum_value 0.01562944527298346
v_order_3 0.015595085142833465
impulse_factor 0.01555518755675827
peak_to_peak 0.01549295097043334
ch3mean 0.015385382950501878
ch6mean 0.01533748368774551
peak_value 0.015314195191304603
cpt1 0.015052180926585345
histogram_upper_bound 0.014692608945198573
max 0.01462061783417942
log_log_ratio 0.014423235165994731
cpt2 0.01419196349385666
crest_factor 0.013996288173871161
square_root_amplitude_value 0.012788205979348695
root_sum_of_squares 0.012508766717371273
clearance_factor 0.012149934636335347
normalized_normal_negative_likelihood 0.012013274302721113
log_rms 0.011385052789774183
sure_entropy 0.010365640805213483
cpt5 0.010154799488905691
std 0.009838552552917873
mean 0.009781623545430827
waveform_indicators 0.009775638406973309
simple_square_integral 0.00914132164725771
absolute_media 0.009104488971902202
energy 0.008481926365376568
root_mean_square 0.008182726668950596
v_order_2 0.008016803132570438
mean_absolute_value_slope 8.879246039390239e-05
average_amplitude_change 5.0638226504777015e-06
difference_absolute_variance_value 3.3727302551445793e-06
slope_sign_change 1.3671817432606714e-06
variance 2.9838190466809033e-07
mean_square_error 0.0
higher_order_temporal_moments 0.0
wilson_amplitude 0.0
myopulse_percentage_rate 0.0
fifth_statistic_moment 0.0
kth_central_moment 0.0
sixth_statistical_moment 0.0
conduction_velocity_signal 0.0
```



2) Second method of feature selection (feature importances with a permutation)

```
# Feature importances with a permutation
result = permutation_importance(rf_classifier, X_train, y_train, n_repeats = 5, scoring = "accuracy", random_state = 191805060, n_jobs = 5)

# Sort permutation importances by mean in descending order
sorted_importances_idx = result.importances_mean.argsort()
importances = pd.DataFrame(result.importances[sorted_importances_idx].T, columns = X.columns[sorted_importances_idx])

ax = importances.plot.box(vert = False, whis = 10)
ax.set_title("Permutation Importances (test set)")
ax.axvline(x = 0, color = "k", linestyle = "--")
ax.set_xlabel("Decrease in accuracy score")
ax.figure.tight_layout()
ax.figure.savefig(f'{main_path}/feature_importances_permutation.png')

# Select the best 10 features from permutation importances
best_features_permutation = importances.mean().nlargest(10).index
print('')
print("Best 10 features from permutation importances:")
print(best_features_permutation)
print('')
```



Find the best 10 features and assign it to “X” variable. ”y” will remain still. Split them again.

```
# Extract the names of the best features
best_feature_names = best_features_permutation.tolist()

# Create X and split data again, keeping only the best features
X = X[best_feature_names]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 50605057)
```

Feature importances from permutation importances:

kurtosis_energy_operator: 0.03845207395796459
zero_crossing: 0.020195176214794494
norm_entropy: 0.006944093307040133
modified_mean_absolute_value_2: 0.0038934115386559886
modified_mean_absolute_value_1: 0.003485530329844444
ch5mean: 0.0010989196204344686
weighted_ssr_absolute: 0.0010854359441101248
kurtosis: 0.0009843083716775114
ch8mean: 0.0008663262038394847
mean_deviation: 0.0006910384116229018

8 Find The Best Model

Define models.

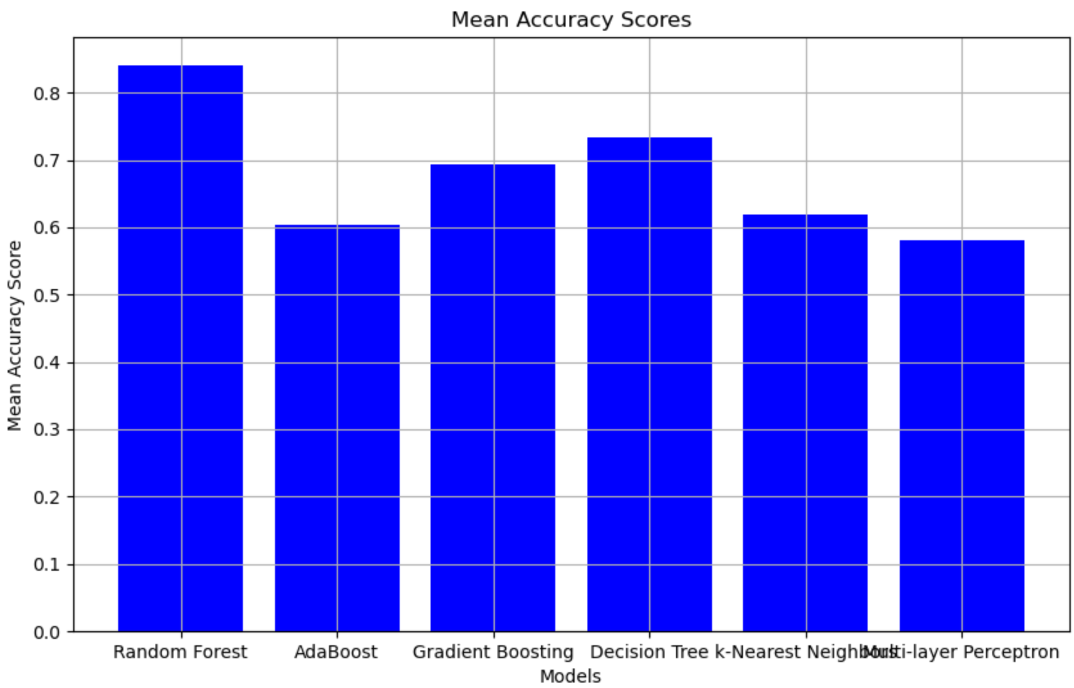
```
def initializeModels():
    return {
        'Random Forest': RandomForestClassifier(),
        'AdaBoost': AdaBoostClassifier(),
        'Gradient Boosting': GradientBoostingClassifier(),
        'Decision Tree': DecisionTreeClassifier(),
        'k-Nearest Neighbors': KNeighborsClassifier(),
        'Multi-Layer Perceptron': MLPClassifier()
    }
```

Calculate mean cross validation scores for each model and find the best model. Save the best model as pickle file.

```python
def crossValidationAndSaveBestModel(models, X_train, y_train):
    results = {}
    acc_scores = []
    model_names = []

    for name, model in models.items():
        scores = cross_val_score(model, X_train, y_train, cv = 10)
        results[model] = scores
        acc_scores.append(np.mean(scores))
        model_names.append(name)

        print('Model:', name, 'Mean Accuracy Scores:', np.mean(scores))

    plotBar('Mean Accuracy Scores', acc_scores, model_names, 'Models', 'Mean Accuracy Score', f'{main_path}/models.png')

    best_model = max(results, key = lambda x: np.mean(results[x]))
    best_model.fit(X_train, y_train)
    with open(best_model_path, "wb") as f:
        pickle.dump(best_model, f)

    return best_model, model_names[np.argmax(acc_scores)]


models = initializeModels()
best_model, best_model_name = crossValidationAndSaveBestModel(models, X_train, y_train)
```

Best model selected as Random Forest and saved as pickle file.

Model	Mean Accuracy Scores
Random Forest	**0.8405555497800942**
AdaBoost	0.6028214795584736
Gradient Boosting	0.6934655131952915
Decision Tree	0.7330063730986353
k-Nearest Neighbors	0.6184288480096046
Multi-layer Perceptron	0.5814151151173499



9 Predict The Gestures

Predict the gestures (classes) in dataset and calculate the accuracy value using the best model.

```
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('')
print(f'Best Model ({best_model_name}) Accuracy:', accuracy)
```

Best Model (Random Forest) Accuracy: 0.8483168161082272

10 Sliding Window

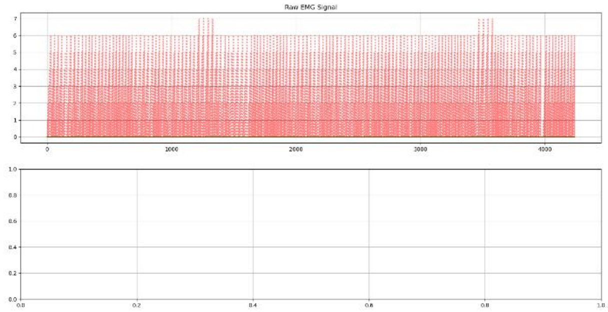
Create keyboard-controlled sliding window. First window displays a time domain plot of raw EMG signals, with the actual label value shown in the title. Second window displays the predicted label value in the title.

```python
# Get data from dataset
folders = [folder.name for folder in os.scandir(dataset_path) if folder.is_dir()]

all_data = pd.DataFrame()

for folder in folders:
    folder_path = os.path.join(dataset_path, folder)
    files = [file.name for file in os.scandir(folder_path) if file.is_file()]
    print(folder, files)
    for file in files:
        file_path = os.path.join(folder_path, file)
        current_data = pd.read_csv(file_path, sep = '\t')
        all_data = pd.concat([all_data, current_data])

rawsignals = all_data.dropna()
rawsignals = rawsignals.to_numpy()

with open(best_model_path, 'rb') as f:
    best_model = pickle.load(f)

features_of_interest = ["kurtosis_energy_operator", "zero_crossing", "norm_entropy", "modified_mean_absolute_value_2",
                        "modified_mean_absolute_value_1", "ch5mean", "weighted_ssr_absolute", "kurtosis", "ch8mean", "mean_deviation"]
```

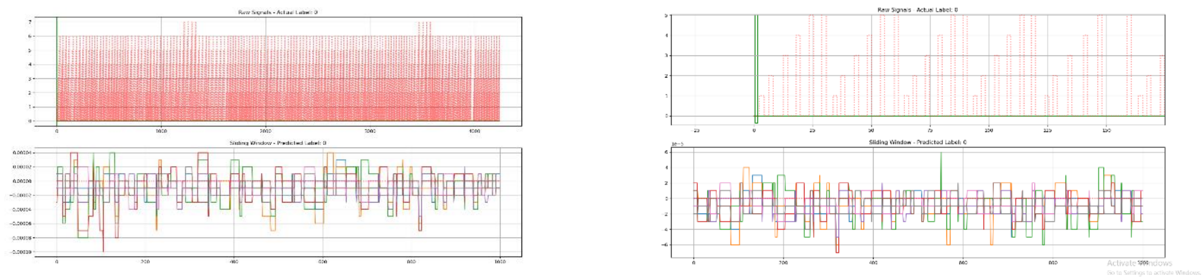
```python
emgch0 = rawsignals[:, 1]
emgch1 = rawsignals[:, 2]
emgch2 = rawsignals[:, 3]
emgch3 = rawsignals[:, 4]
emgch4 = rawsignals[:, 5]
emgch5 = rawsignals[:, 6]
emgch6 = rawsignals[:, 7]
emgch7 = rawsignals[:, 8]

outclass = rawsignals[:, 9]
lensignal = len(emgch0)
fs = 1000
time = np.arange(lensignal) / fs

winsize = 1000
winhop = 500
i = 0

def on_press(event):
    global i
    sys.stdout.flush()

    lower = i
    upper = i + winsize

    ax1.cla()
    ax1.plot(time, emgch0, 'g')
    ax1.plot(time, outclass, 'r:', linewidth=2, alpha=0.5)
    ax1.grid()
    ax1.set_title(f'Raw Signals - Actual Label: {int(outclass[lower])}')

    ax1.add_patch(patches.Rectangle((time[lower], ax1.get_ylim()[0]), winsize/fs, ax1.get_ylim()[1] - ax1.get_ylim()[0], linewidth=2, edgecolor='g', facecolor='none'))



    selmat = all_data.iloc[lower:upper, 1:-1].to_numpy().flatten()
    features_dict  = extractFeatures(selmat)
    ch_means = all_data.iloc[lower:upper, 1:9].mean().to_numpy()
    features_dict.update({
        f'ch{i}mean': ch_means[i-1] for i in range(1, 9)
    })
```

```python
    ch_means = all_data.iloc[lower:upper, 1:9].mean().to_numpy()
    features_dict.update({
        f'ch{i}mean': ch_means[i-1] for i in range(1, 9)
    })

    selected_features = [features_dict[feature] for feature in features_of_interest]
    selected_features_df = pd.DataFrame([selected_features], columns=features_of_interest)


    y = outclass[lower:upper]

    ax2.cla()
    for x_slice in [emgch0[lower:upper], emgch1[lower:upper], emgch2[lower:upper], emgch3[lower:upper], emgch4[lower:upper], emgch5[lower:upper], emgch6[lower:upper], emgch7[lower
        ax2.plot(x_slice)

    # Predict using the best model
    predicted_label = best_model.predict(selected_features_df)

    ax2.plot(y, 'r:', linewidth=2, alpha=0.5)
    ax2.grid()
    ax2.set_title(f'Sliding Window - Predicted Label: {int(predicted_label)}')

    if event.key == 'right':
        i += winhop
        fig.canvas.draw()
    elif event.key == 'left':
        i -= winhop
        fig.canvas.draw()

fig = plt.figure()
ax1 = fig.add_subplot(211)
ax1.plot(time, emgch0, 'g')
ax1.plot(time, outclass, 'r:', linewidth=2, alpha=0.5)
ax1.grid()
ax1.set_title('Raw EMG Signal')

ax2 = fig.add_subplot(212)
ax2.grid()

fig.canvas.mpl_connect('key_press_event', on_press)

plt.show()
```

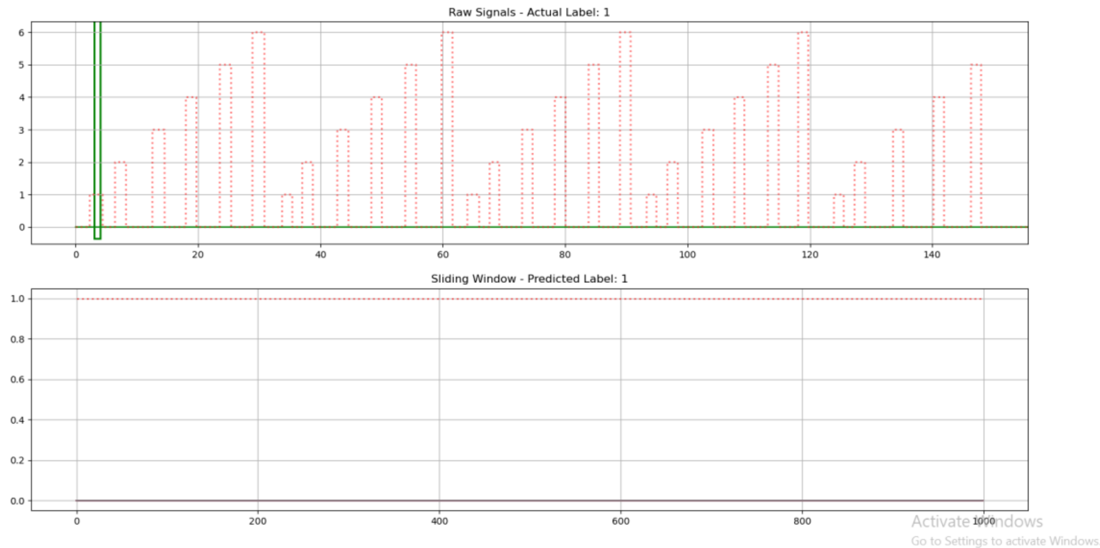
Sliding window 0. stride:



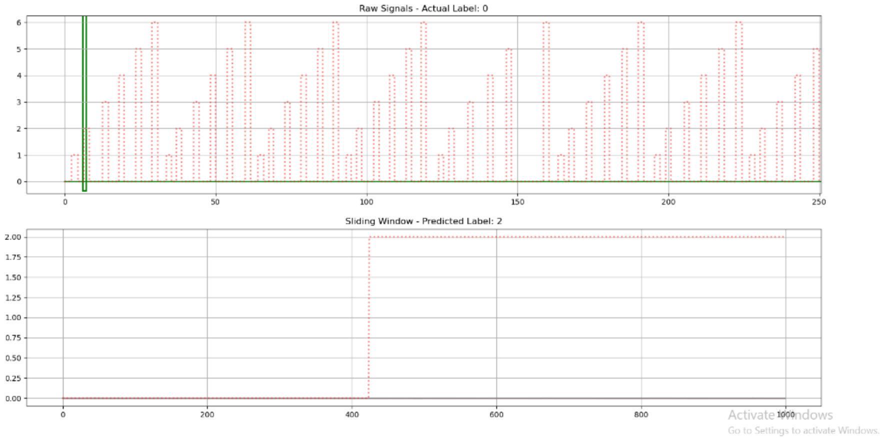
Sliding window 1. stride and zoomed in version:



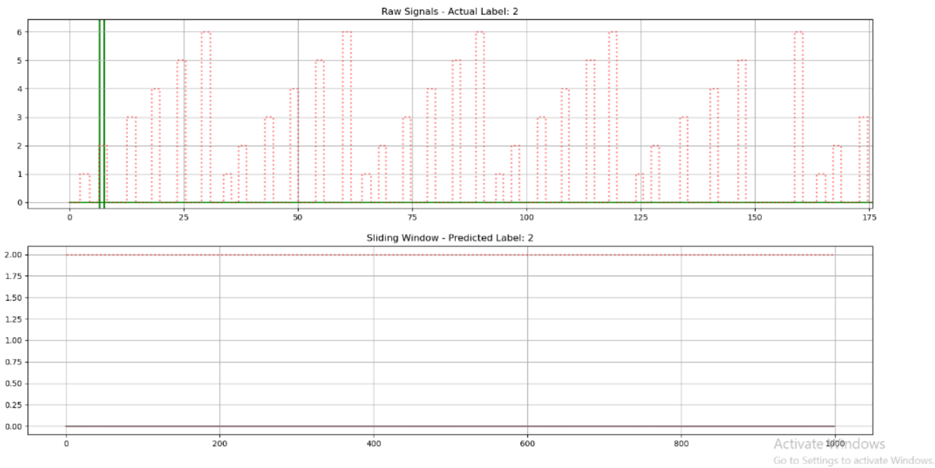
Sliding window 6. stride with correct prediction:



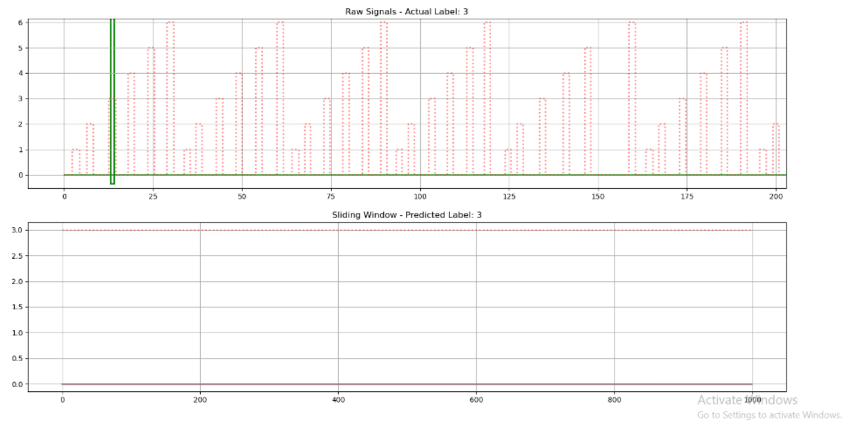
Sliding window 13. stride with wrong prediction:



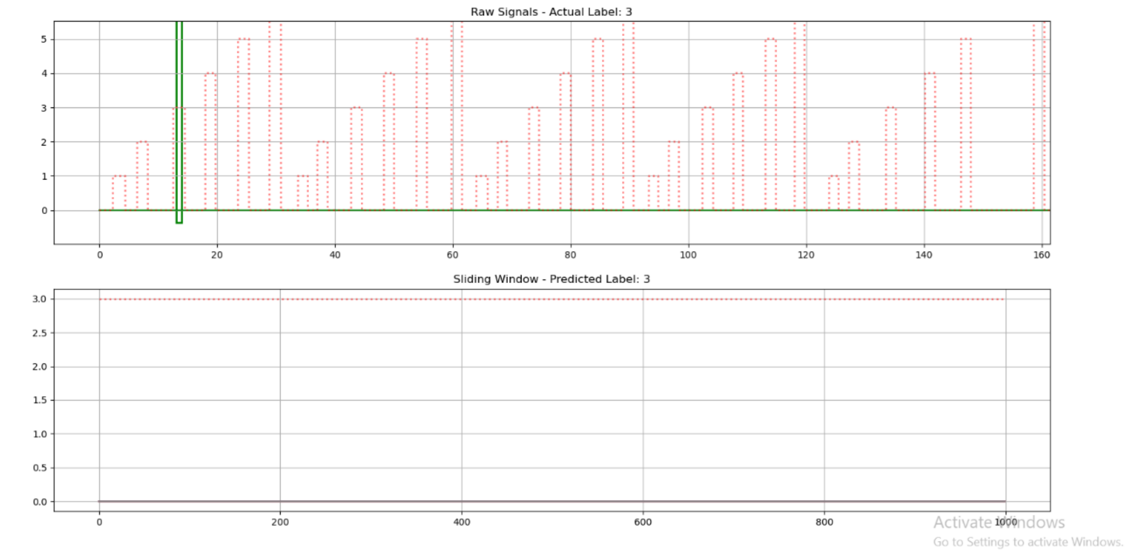
Sliding window 14. stride with correct prediction:



Sliding window 26. stride with wrong prediction:



Sliding window 27. stride with correct prediction:



Sliding window 31. stride with correct prediction