EMG Analysis
Project Report
Digital Signal Processing
Fall 2024
The American University in Cairo

Presented to: Dr. Seif Eldawlatly
Aly Elaswad 900225517
Mohamed Alaa 900212213
Omar Ganna 900222646

1. Project description and the approach used

Project Description

In this project, we were given the datasets of three different subjects, and for each subject, we were given the following:

1. Subject number
2. sEMG signal, containing electrodes equally spaced around the forearm at the height of the radio humeral joint and signals from the main activity spot of the muscles flexor and extensor digitorum superficialis.
3. Stimulus, which is the movement repeated by the subject according to the displayed move

All subjects were asked to perform two hand movements and rest between them and the two hand movements were abduction of all fingers and flexing the fingers together in fist.

It was also given that the sampling rate of the data is 100 Hz.

Approach taken

1. High-pass Filter

As a pre-processing, we implemented a high-pass filter with a cutoff frequency of 10 Hz using the `butter` and `lfilter` functions in the scipy library, below is a code snippet of how it was done:

```
def highpass_filter(data, cutoff=10, fs=100, order=4):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    filtered_data = lfilter(b, a, data, axis=0)
    return filtered_data
```

Here, the function takes as inputs

1. The data to be filtered
2. The cutoff frequency (in our case it was 10 Hz)
3. The sampling rate F_s (100 Hz for this data)
4. The order determines the steepness of the transition band and is required by the butter function

The Nyquist rate is first calculated, which is half the sampling rate since the Nyquist theorem states that:

$$F_s \geq 2F_{max}$$

The cutoff frequency is then normalized with respect to the Nyquist frequency.

Then, a Butterworth filter is applied and the returned coefficients `a` and `b` are given to our data using the `lfilter` function, which applies a linear high-pass filter to the signal.

---

2. Feature Extraction for Method 1 (Time-Domain Analysis):

The sEMG signals were segmented into trials, with each trial representing a specific hand movement stimulus. The basis for the segmentation was the stimulus signal to ensure that each trial's data contained only one distinct movement. This is one of the challenges we faced in this project, where we had to find a way to segment the data which was of unequal lengths, we tried two approaches, the first one being to choose the signal with the minimum length and limit this range to be our length. As another approach, we truncated the signal to a fixed number of samples to eliminate the variability between the different trials.

Below is the code showing our implementation for this method

```
def Extraxt_segmented_trials(emg, stimulus, fs, trial_duration=5, break_label=0):
    trial_changes = np.where(np.diff(stimulus) != 0)[0] + 1
    trial_starts = np.concatenate(([0], trial_changes))
    trial_ends = np.concatenate((trial_changes, [len(stimulus)]))


    trials = []
    labels = []
    trial_samples = trial_duration * fs
    for start, end in zip(trial_starts, trial_ends):
        trial_data = emg[start:end, :]
        trial_label = stimulus[start]
        if trial_label == break_label:
            continue
        if len(trial_data) >= trial_samples:
            trial_data = trial_data[:trial_samples]
            trials.append(trial_data.flatten())
            labels.append(trial_label)
    return trials, labels
```

Here, it can be seen that we ignored the “at rest” stimuli by using `break_label` where we do not append the zero stimuli to the trials using the `continue` statement.

It can also be seen that the raw data is the one being extracted in the trials, and we also flatten the data into a 1D array to be given to the KNN classifier that expects vectorized inputs.

3. Feature Extraction for Method 2 (Frequency-Domain Analysis):

As for the frequency domain analysis, we convert the signals into the frequency domain signal using the Fast Fourier Transform and we only consider half the signal (up to the Nyquist rate), since the Fourier transform is symmetric for real-valued signals, and redundant information is at the second half of the spectrum, we also flatten the data just like we did in the time-domain analysis. Below is the code snippet of how this was implemented:

```
def extract_frequency_features(signal):
    fft_values = np.fft.fft(signal, axis=0)
    fft_magnitude = np.abs(fft_values)
    half_idx = len(fft_magnitude) // 2
    fft_magnitude = fft_magnitude[:half_idx]
    return fft_magnitude.flatten()
```

4. Combining features from both methods (Method 3):

For method 3, we **concatenated** either the time-domain features or the frequency-domain features based on the input, we also tried to concatenate them both together for testing purposes. Below is the code snippet of this method's implementation:

```
def extract_combined_features(trial, feature_type='combined'):
    if feature_type == 'time':
        features = np.concatenate([extract_time_features(trial[:, i]) for i in range(trial.shape[1])])
    elif feature_type == 'frequency':
        features = np.concatenate([extract_frequency_features(trial[:, i]) for i in range(trial.shape[1])])
    else:  # For testing purposes
        time_features = np.concatenate([extract_time_features(trial[:, i]) for i in range(trial.shape[1])])
        freq_features = np.concatenate([extract_frequency_features(trial[:, i]) for i in range(trial.shape[1])])
        features = np.concatenate([time_features, freq_features])

    return features
```

5. Leave-one-trial-out approach for the KNN classifier

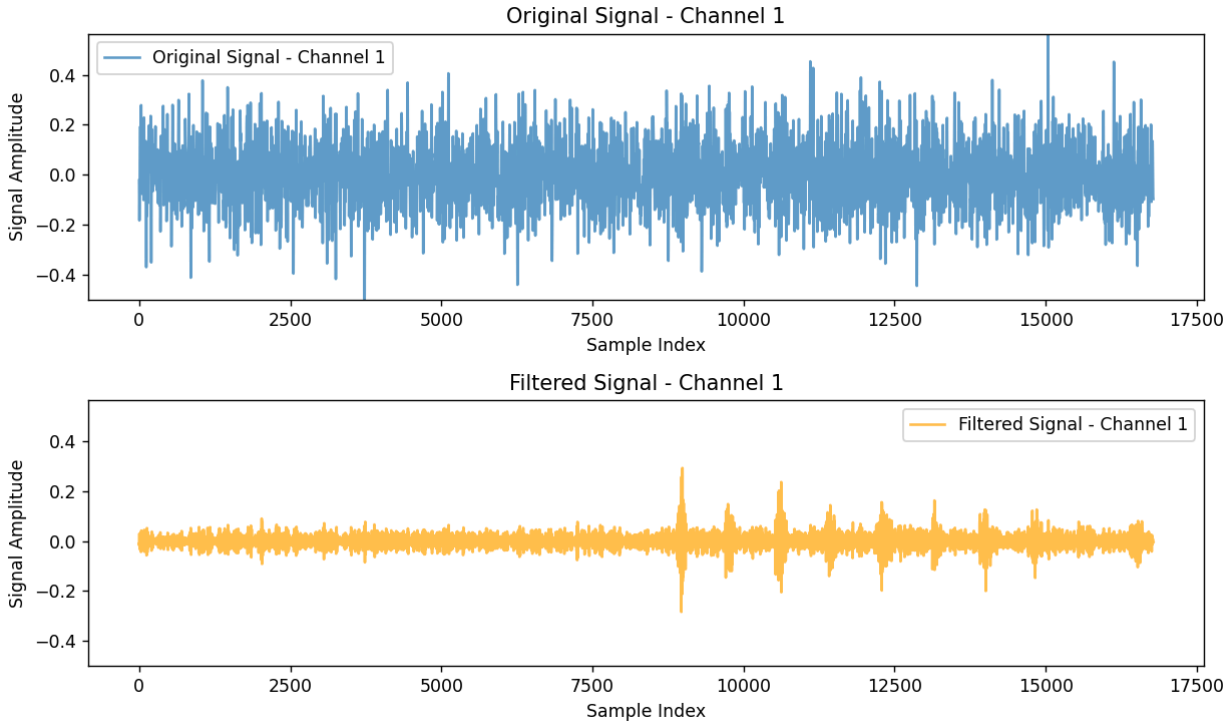
The leave-one-trial-out approach was the one used for evaluation. Each trial was used as the test set exactly once and we used a variable called `train_mask` to exclude the current test trial index when creating the training set, which ensures that the model is being evaluated on data that was not part of its training set. We apply **feature scaling** to the data using standardization to ensure that all features contribute equally to the distance metric used by the KNN. Below is the code snippet of how this was done:

```python
def leave_one_trial_out_cv_combined(trials, labels, k_range, feature_type='combined'):
    n_trials = len(trials)
    k_scores = {k: [] for k in k_range}

    X = np.array([extract_combined_features(trial, feature_type) for trial in trials]) # changes based on method
    y = np.array(labels)

    for k in k_range:
        trial_predictions = []
        trial_true_labels = []

        for test_idx in range(n_trials):
            train_mask = np.ones(n_trials, dtype=bool)
            train_mask[test_idx] = False

            X_train = X[train_mask]
            y_train = y[train_mask]
            X_test = X[~train_mask]
            y_test = y[~train_mask]

            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train)
            X_test_scaled = scaler.transform(X_test)

            k_adjusted = min(k, len(X_train))

            knn = KNeighborsClassifier(n_neighbors=k_adjusted)
            knn.fit(X_train_scaled, y_train)
            pred = knn.predict(X_test_scaled)

            trial_predictions.extend(pred)
            trial_true_labels.extend(y_test)

        accuracy = accuracy_score(trial_true_labels, trial_predictions)
        k_scores[k].append(accuracy)
        print(f"  - K: {k}, Accuracy: {accuracy:.4f}")

    return k_scores
```

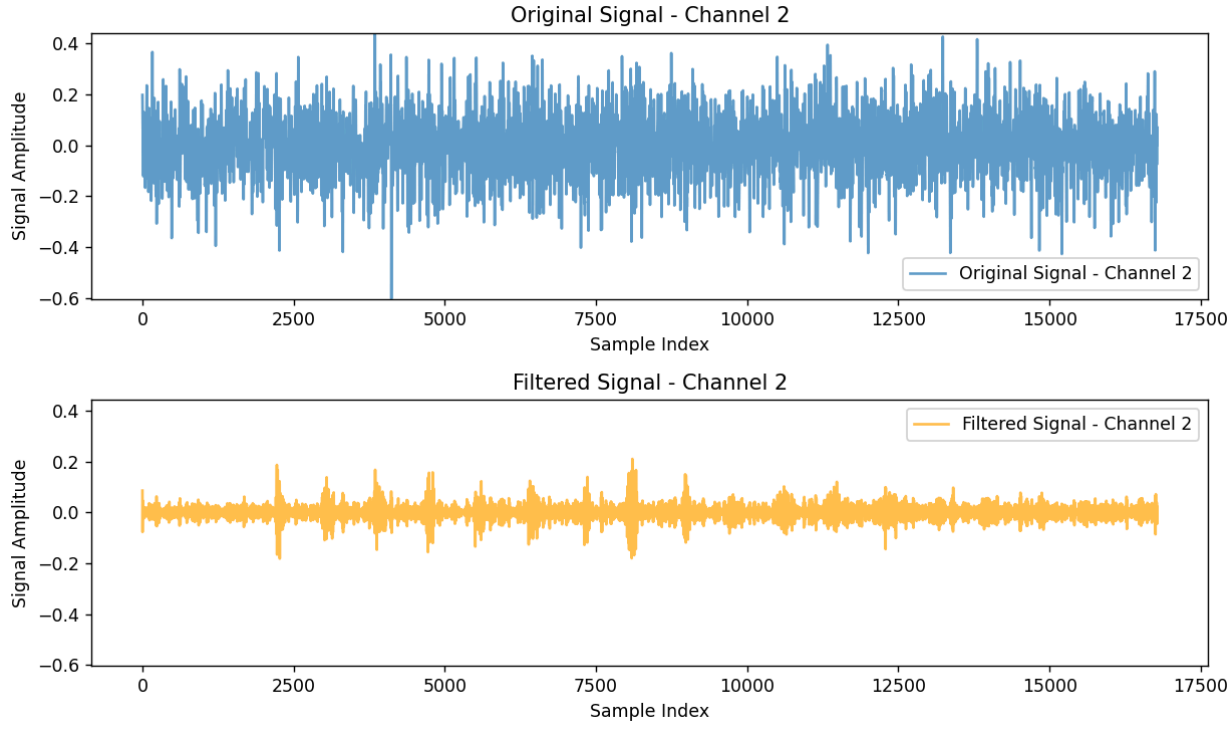
2. Outputs of the project

- Time domain plots

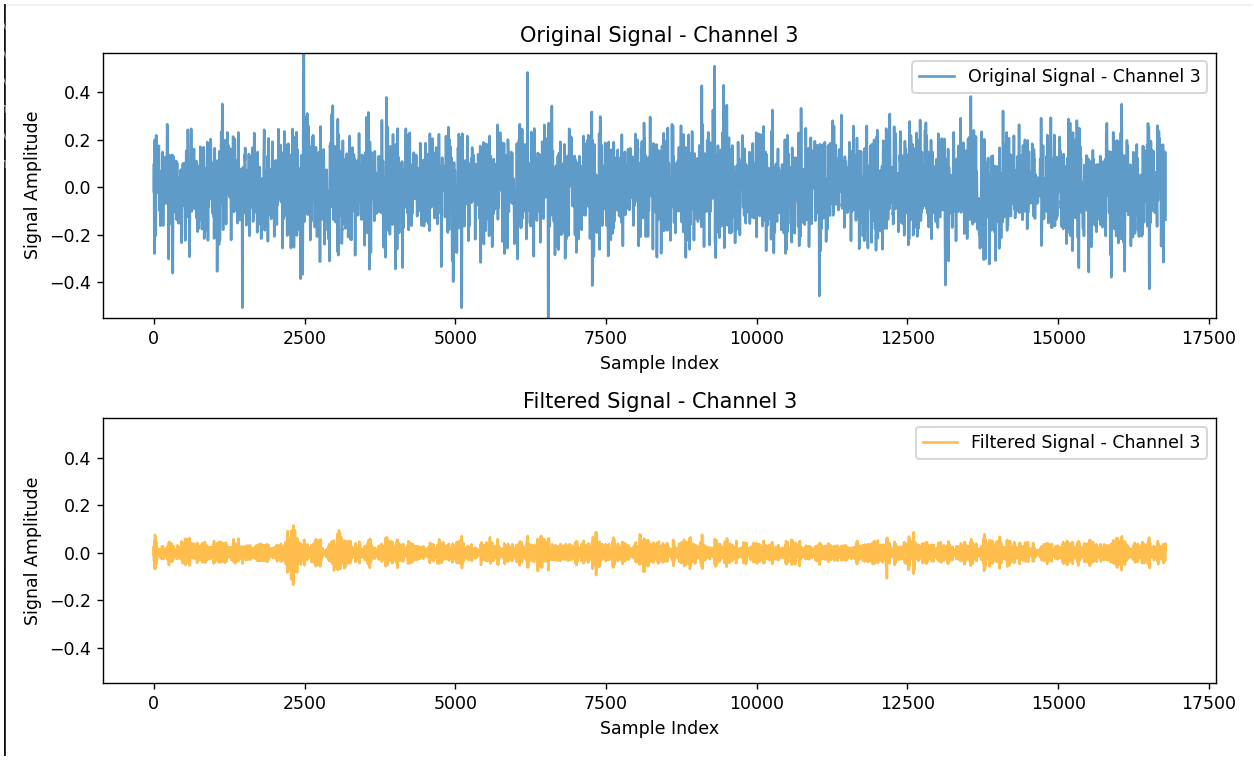
Plot of the first channel before and after applying the high pass filter:



Plot of the second channel before and after applying the high pass filter:

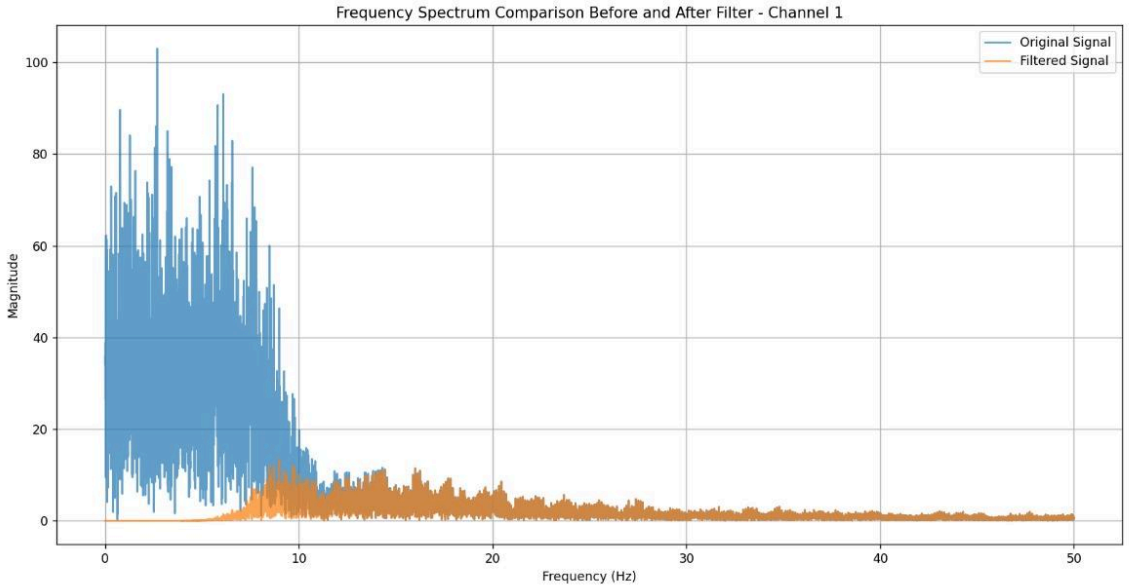


Plot of the third channel before and after applying the high pass filter:

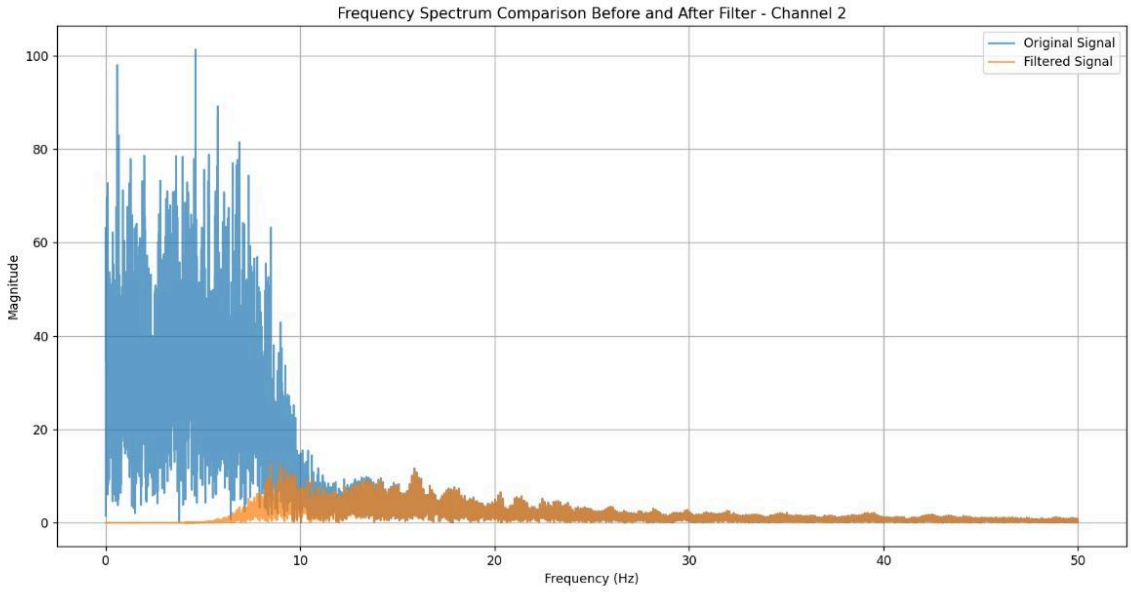


- Frequency Domain plots

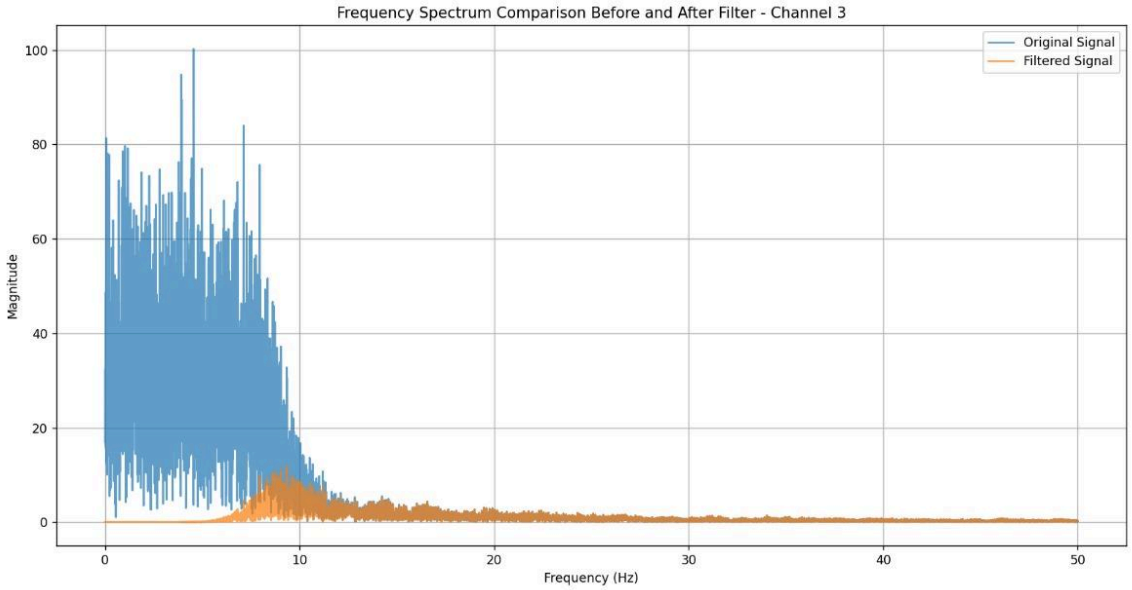
Plot of the first channel before and after applying the high pass filter:



Plot of the second channel before and after applying the high pass filter:



Plot of the third channel before and after applying the high pass filter:



Commenting on the impact of the high-pass filter:

As for the time domain plots, it can be seen that the filtered signal shows a much cleaner waveform with reduced low-frequency content. The signal is less dominated by slow, drifting components, which indicates that the low-frequency noise has been successfully suppressed as desired, whereas higher-frequency details are preserved.

As for the frequency domain plots, it can be seen that in the original signal, there was a lot of energy in the lower frequencies, whereas in the filtered signal, most of the low-frequency components are attenuated, which shows that the filter correctly removes low-frequency components.

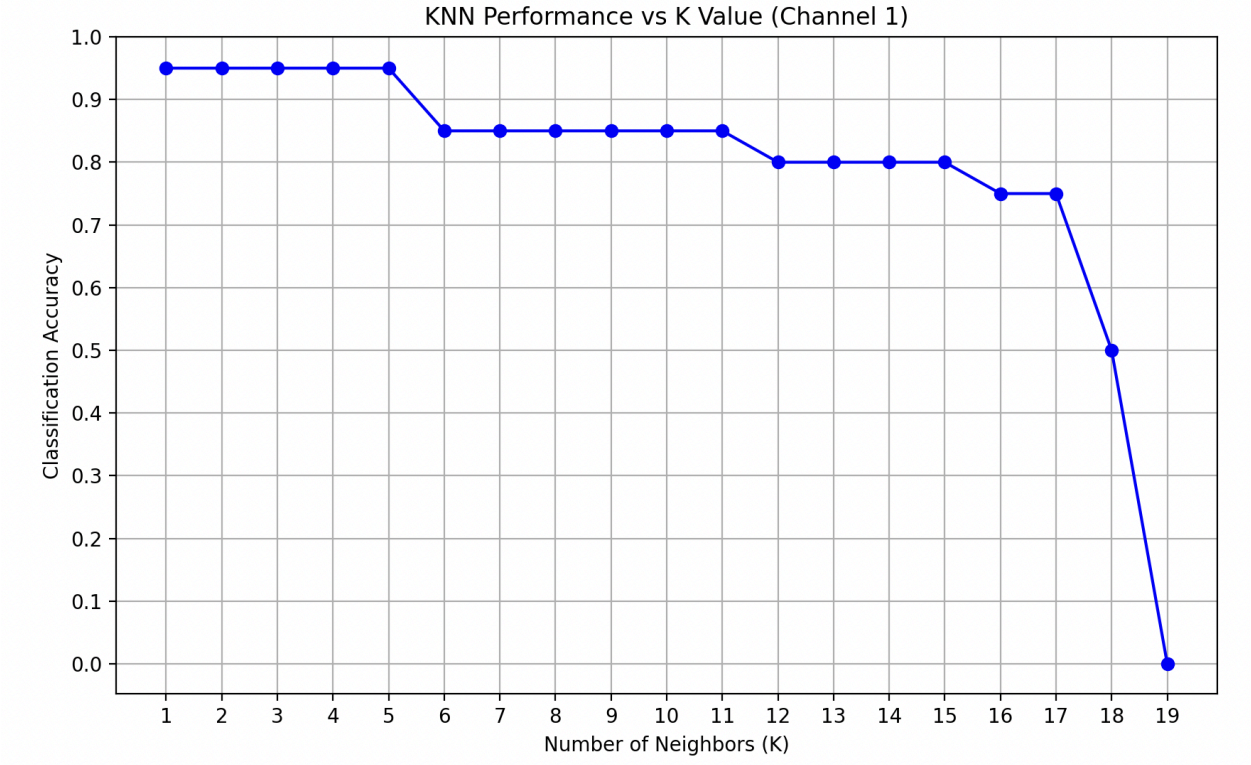
- **Leave-one-out classification accuracy**

We did two approaches in segmenting the trials, one of them is by determining the trial with the minimum length and making this our signal length and the other approach was by a fixed length irrespective of the variable data length. Below are the results:

❖ For the minimum length approach

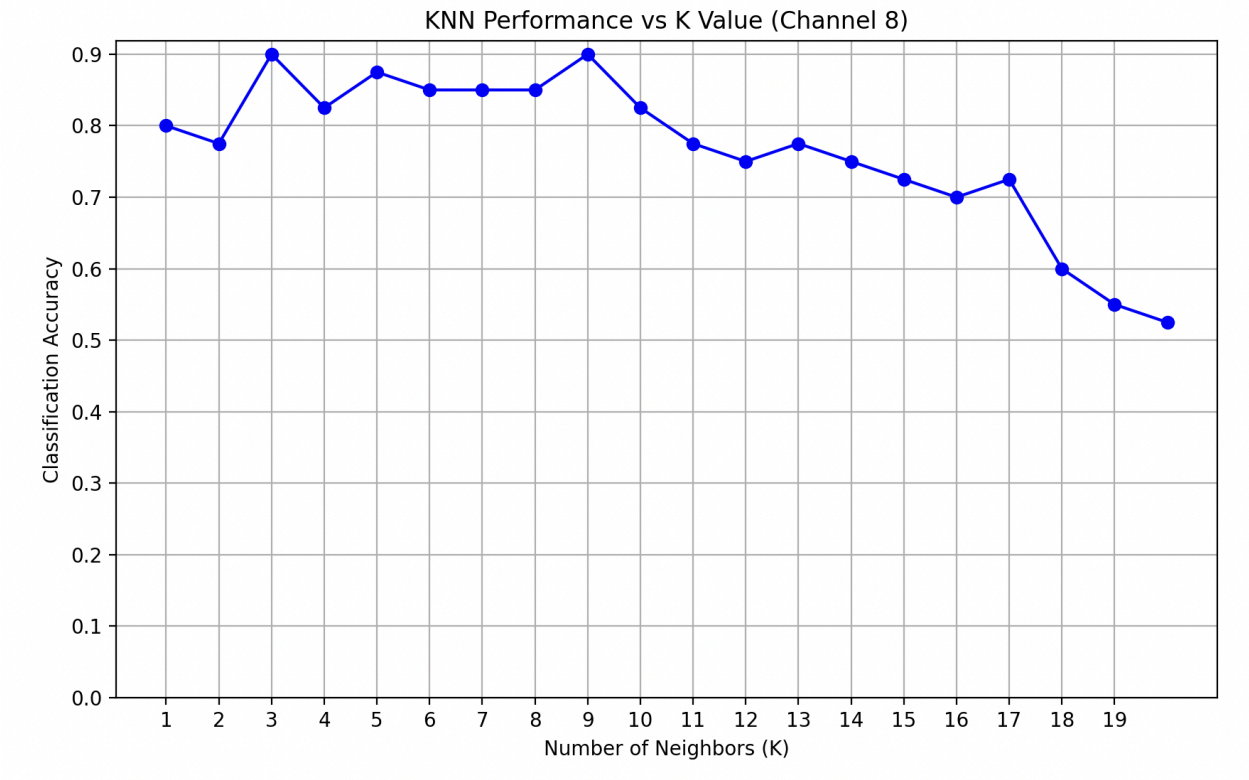
For **subject 1**, the **frequency domain** had the highest leave-one-out classification accuracy of 95% in channel 1, and for the value of ***K*** equals 1.

The (Accuracy vs K) plot of this channel can be seen here:



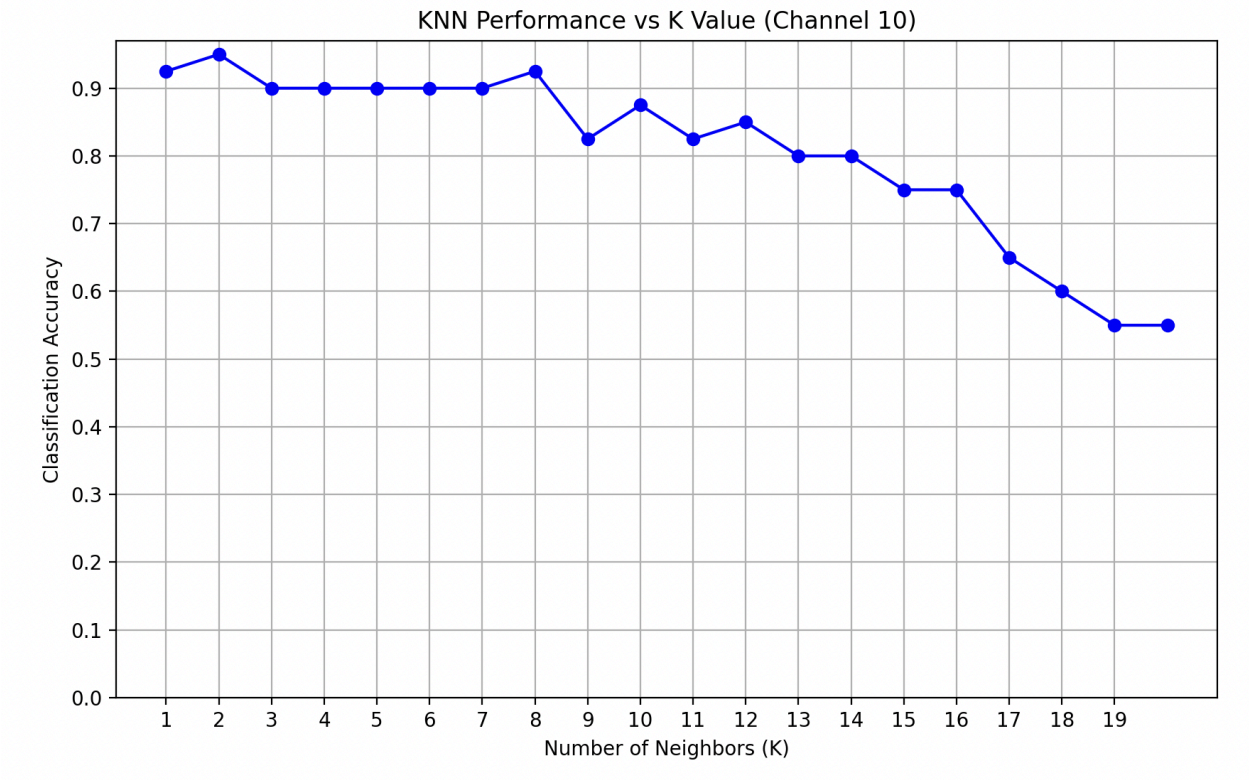
For **subject 2**, the **frequency domain** had the highest leave-one-out classification accuracy of 90% in channel 8, and for the value of ***K*** equals 3.

The (Accuracy vs K) plot of this channel can be seen here:



For **subject 3**, the **frequency domain** had the highest leave-one-out classification accuracy of 95% in channel 10, and for the value of ***K*** equals 2.

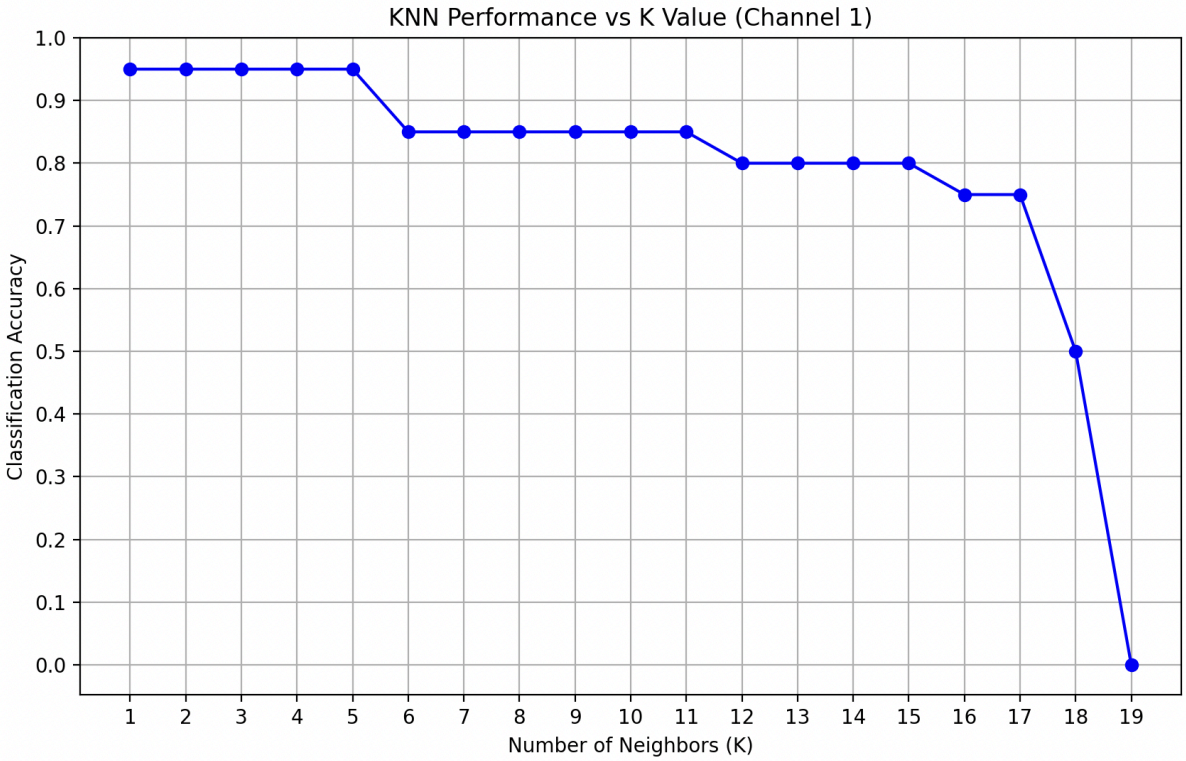
The (Accuracy vs K) plot of this channel can be seen here:



- For the fixed-length approach

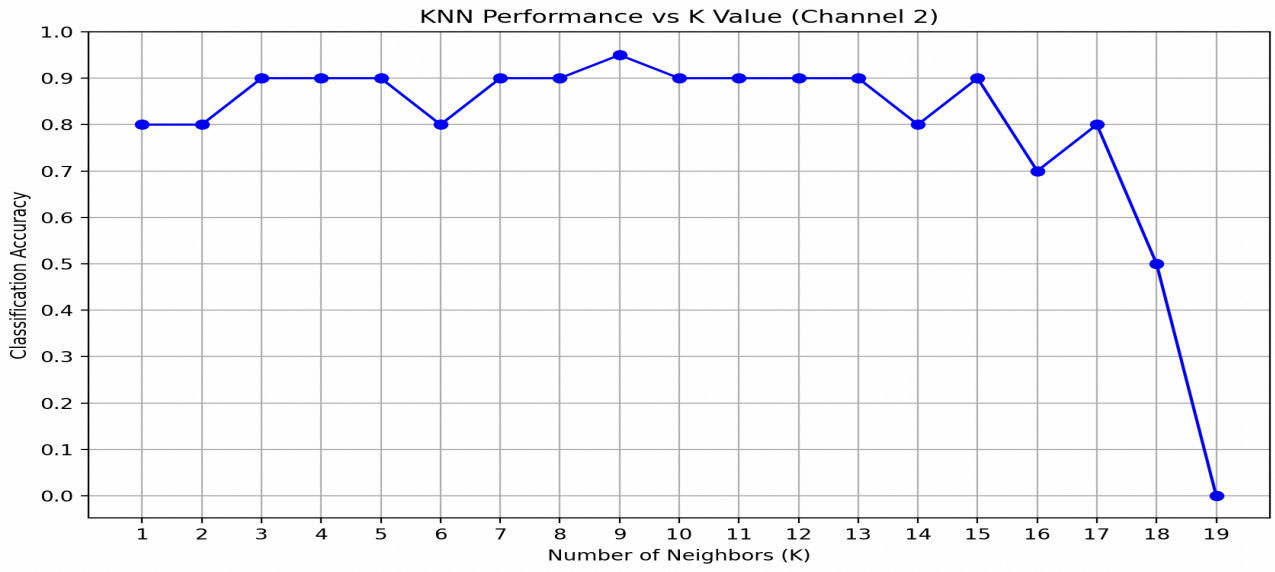
For **subject 1**, the **frequency domain** had the highest leave-one-out classification accuracy of 95% in channel 1, and for the value of ***K*** equals 1.

The (Accuracy vs K) plot of this channel can be seen here:



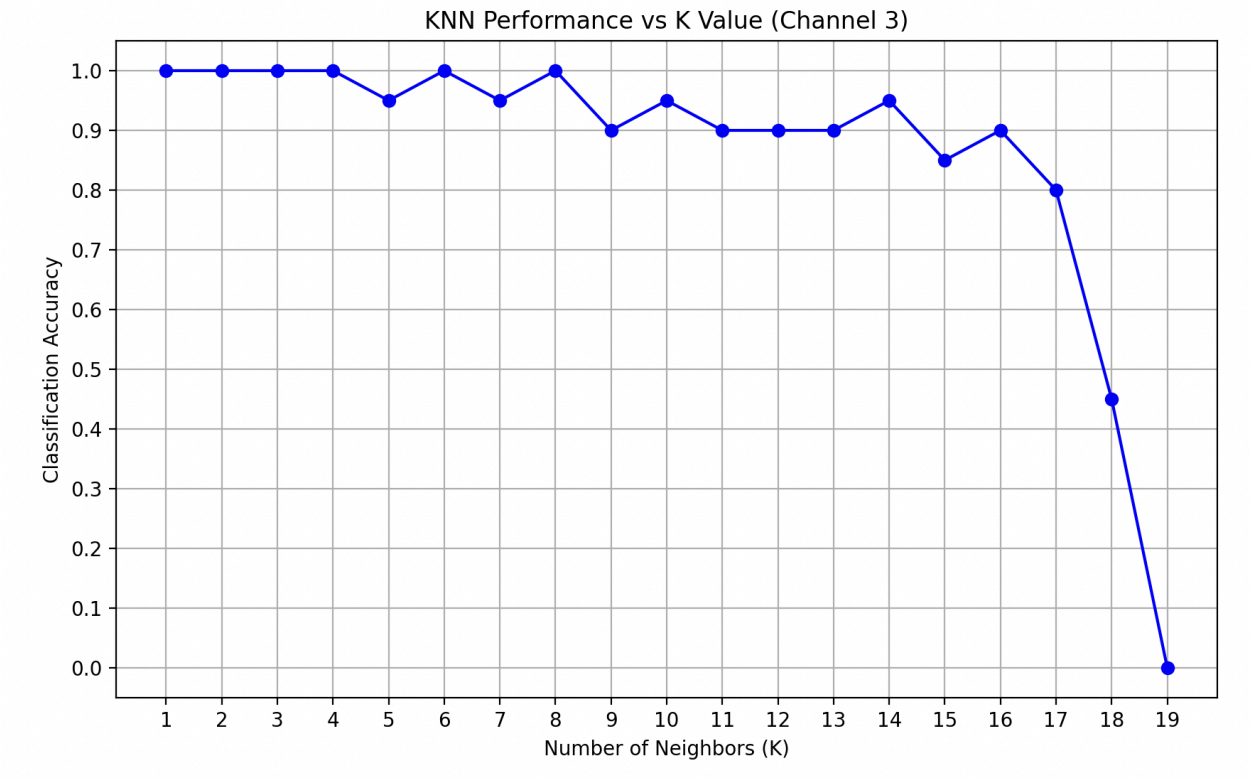
For **subject 2**, the **frequency domain** had the highest leave-one-out classification accuracy of 95% in channel 2, and for the value of ***K*** equals 9.

The (Accuracy vs K) plot of this channel can be seen here:



For **subject 3**, the **frequency domain** had the highest leave-one-out classification accuracy of 100% in channel 3, and for the value of ***K*** equals 1.

The (Accuracy vs K) plot of this channel can be seen here:



Comments on the differences across the three subjects:

Across the three subjects, it can be seen that the accuracy in the **frequency domain** is consistently better than the accuracy in the time domain.

We can also see that in many cases, lower values of K give better KNN performance relative to higher values of K in the same channel.

As for the different channels, there was no clear pattern of certain channels getting better performances than others.

Combining the data from all channels:

To combine the data from all channels, we concatenated features from all channels into a single feature vector, but we first extracted features separately for each channel, to ensure that the unique characteristics of each signal are preserved, which respects the spatial and temporal variability in multi-channel data. This operation preserves the unique contribution of each channel while combining them into a single representation for subsequent analysis. We then had three different approaches to concatenate them:

1. Time-domain features

   Where we concatenated the **time-domain** data from all channels

2. Frequency-domain features

   Where we concatenated the **frequency-domain** data from all channels

3. Combining frequency-domain with time-domain features

   Once features are extracted for each channel, they are concatenated.

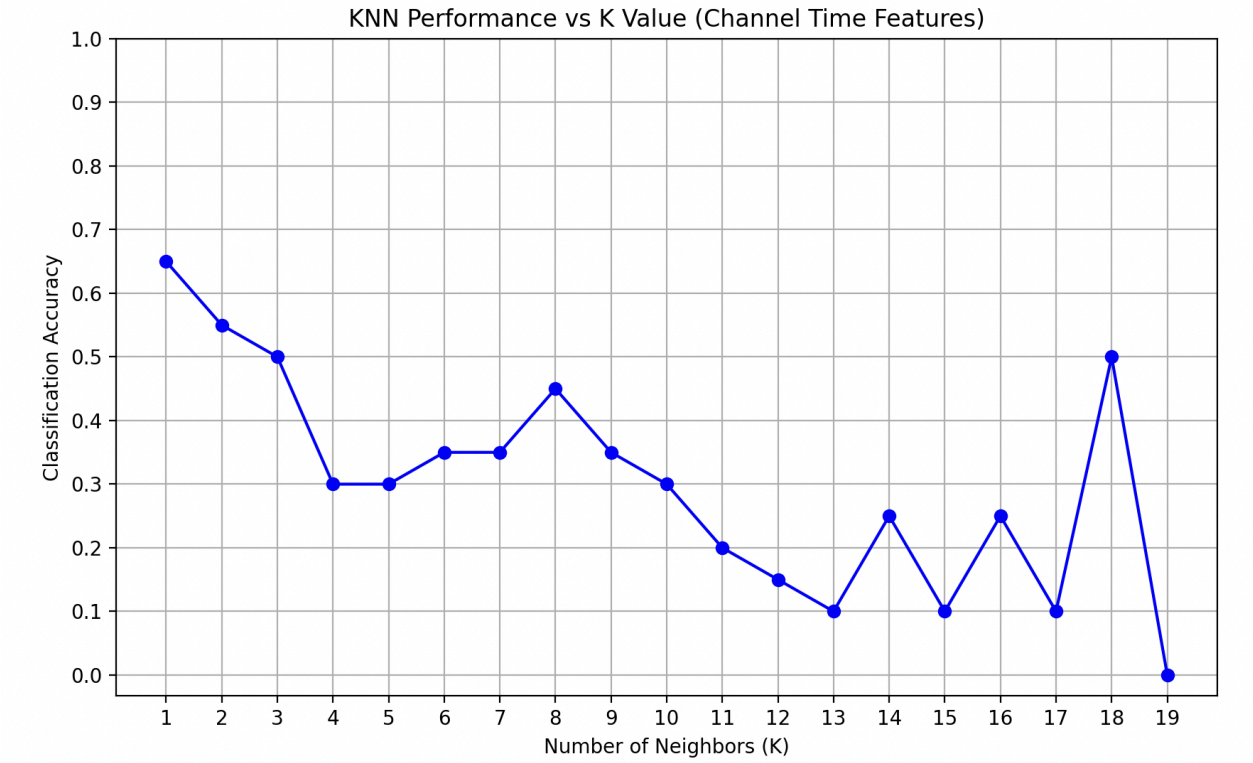
   This operation stacks the feature vectors for all channels into one single, large feature vector.

The impact of considering all channels (Method 3):

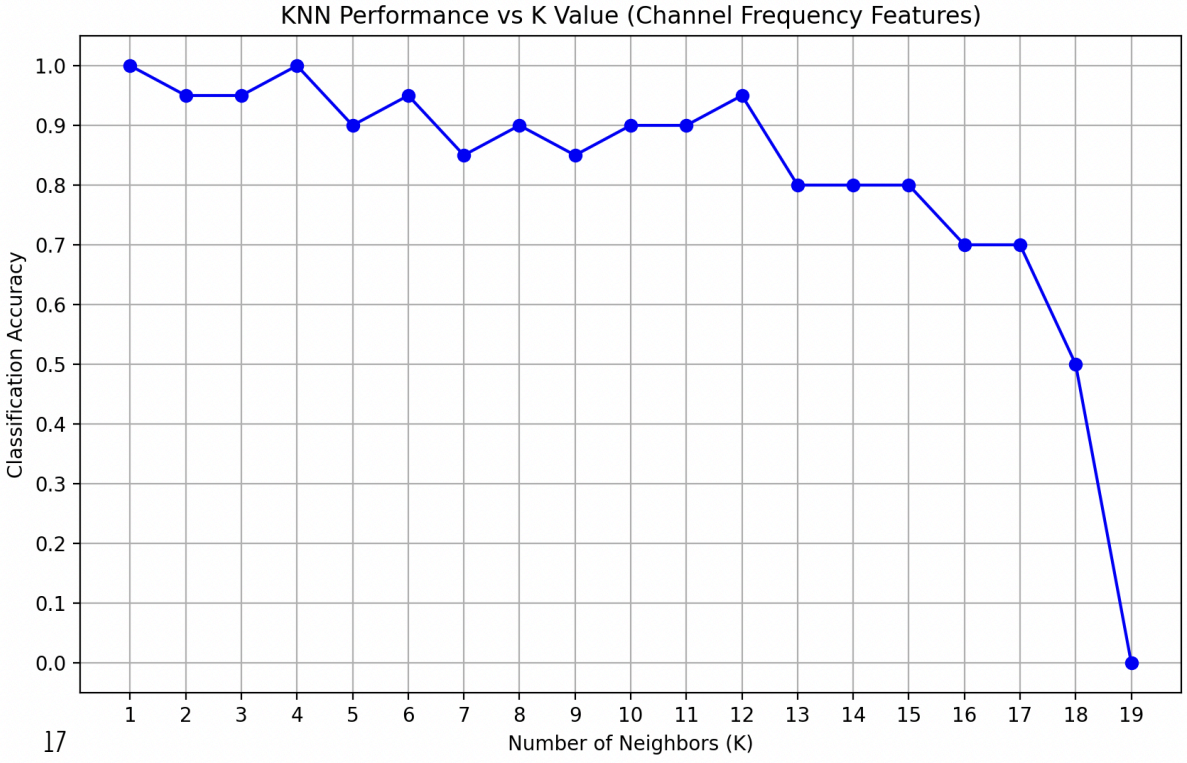
- When concatenating all channels, we found that our first approach, where we concatenated the time-domain data from all channels, resulted in worse accuracy.
- Whereas for the second approach, where we concatenated the frequency-domain data from all channels, we found that accuracy was close to the accuracy of the data from each independent channel.
- For the third approach, concatenating them both together, we also found that the accuracy was fluctuating around the values obtained in methods 1 and 2.

Below is a plot of each of the three approaches for subject 1 as an example:

Concatenating Time features across all channels:



Concatenating Frequency features across all channels:



Concatenating Frequency features with time features across all channels: