Classification of neuron types based on extracellular potential spikes

1. Introduction

Classification is a strong point of computation, that finds widespread application in a multitude of situations in today's world. This paper will explore two different types of classification methods for the correct labelling of extracellular potential spikes based on five neuron types. The two methods are K nearest neighbour and a multilayer perceptron.

2. Performance Metrics

There are many different methods of assessing the performance of a classification algorithm. The first, and most simple, is accuracy. Accuracy simply uses the number of correctly classified data points and the number of data points overall to produce a percentage value of performance. This is very useful when, for example, using algorithmic optimisation. This is due to the simplicity of calculation and high relevance to the improvement of the classification method. Four other key indicators of a methods strengths and weaknesses. True positives, False positives, True negatives and false negatives are also extremely applicable. These indicators are used throughout the development of all the methods used in this report, especially in the development of the peak detection algorithm, as they provided valuable feedback for development and adjustment of the designs [1]. The final performance metric based on these four key parameters is the confusion matrix. A confusion matrix visualises misclassification of data points providing useful insight into the behaviour of the classification method.

3. Classification

The two methods chosen for this problem are a multilayer perceptron and K-nearest neighbour. The initial steps of classification use very similar initial steps as the pre-processing is not required to change. *Figure 1* show a system diagram of the two methods and how they are intertwined.

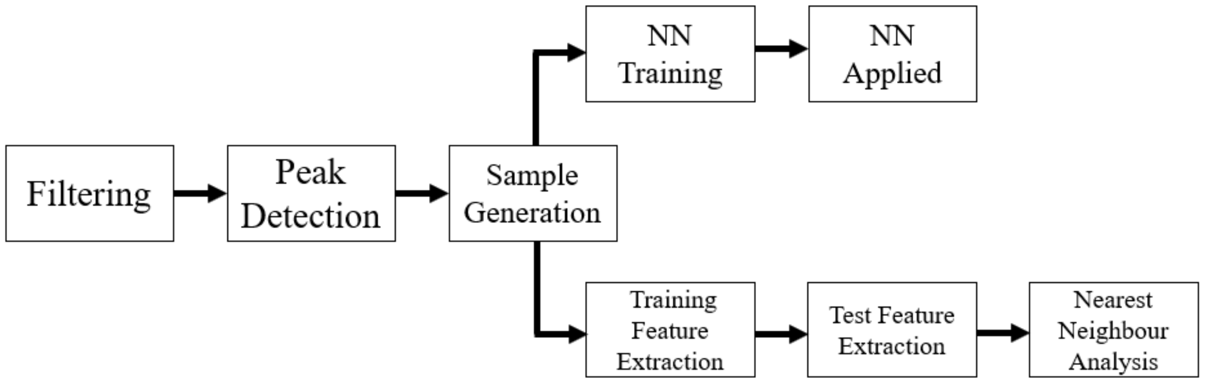


Figure 1: Flow diagram of K-nearest neighbour and multilayer perceptron

3.1 Input Preparation

3.1.1 Signal Pre-Processing

The first step in designing the multilayer perceptron (MLP) is assessing the given data. For the MLP to have the highest chance of success, it must be fed the highest quality data to learn from.

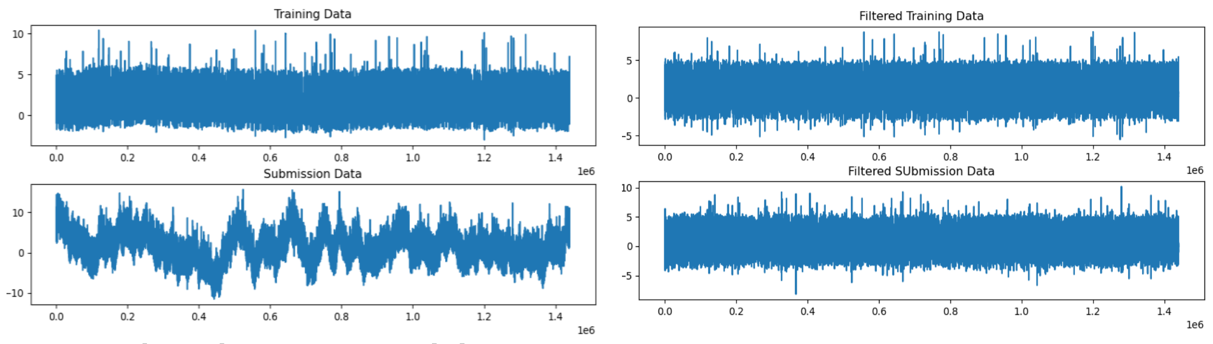


Figure 2*a* shows the ı A ata provided. B

Figure 2: [A] Raw data, [B] Filtered data

While both datasets contain a large amount of high frequency noise, the submission data contains significant low frequency noise. This is removed using a band pass filter. Initial parameters for this filter were set to 25Hz-5000Hz and the results can be seen in

Figure 2*b*.

3.1.2 Peak Detection

Detecting the peaks in the data enables the production of normalised data sets that can be fed to the neural net spike by spike in order to train and test it. The first step of producing these widows is detecting where the peaks are located. While some simpler methods such as threshold detectors could be viable, the find_peaks function from the scipy library offers a simple yet powerful alternative. The default settings result in the overdetection of peaks shown in *Figure 3*.

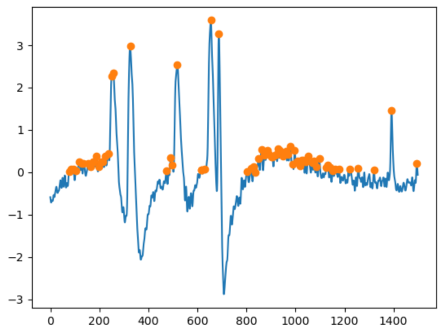


Figure 3: Default peak detection using scipy's find_peaks

By varying two key parameters (height and prominence) the function can be tweaked to detect only the desired peaks, denoted by the index data provided. Initially, the altering of these variables was done through visual analysis of the peaks.

The indexes used in the training dataset tend to be placed roughly at the start of the peak, therefore, the deduced peak locations were shifted to match. The first minima before the peak were used as the definition for start of the peak.

Once the function was working to a reasonable level, a method was developed for counting missed peaks, false positives and correct peaks using the index values provided. This was incredibly valuable as it acted as a performance metric providing and indicator for further development. *Figure 4* shows the breakdown and display of the peak detector's efficacy.

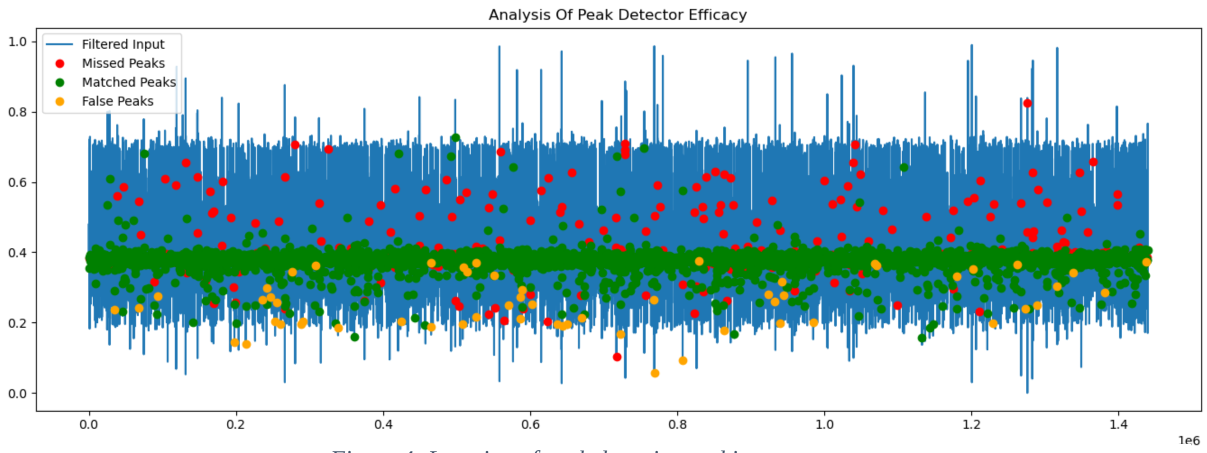


Figure 4: Location of peak detection and its accuracy

Small modifiers were added to the detection to adjust for certain conditions. The first example of this is the removal of false positives on rebound peaks after a spike due to a neuron firing shown in *Figure 5*.

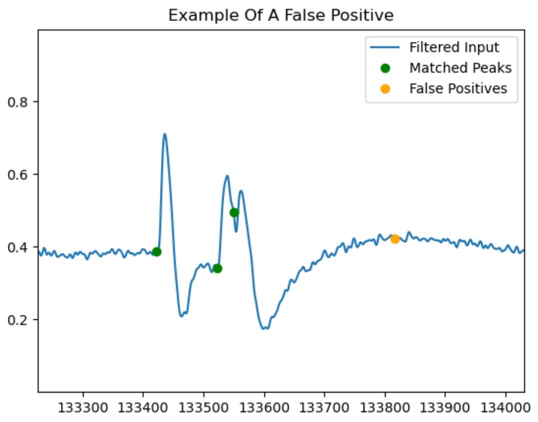


Figure 5: False positive from peak detector

The false peaks were removed by measuring the width of the peaks. The false peals had a significantly wider peak making them differentiable from the correct peaks by setting a limit on the allowable width. Another issue encountered was the locating of the labels at the start of a peak. If a peak was not 'sharp' and had a minimum at the peak, the label would not be correctly moved. This was remedied by adding a required drop in height from the peak before the label could be moved.

3.1.3 Performance

The final performance of the peak detection is shown below in *Table 1*.

Table 1: Peak detector performance

Correctly labelled peaks	2996
Missed peaks	346
False positives	0

3.1.3 Sample Preparation

Producing the final input frame for the neural network can now take place as the peaks and their locations have been calculated. *Figure 6* Shows the entire set of peaks provided in the training data, with the mean shown in black.

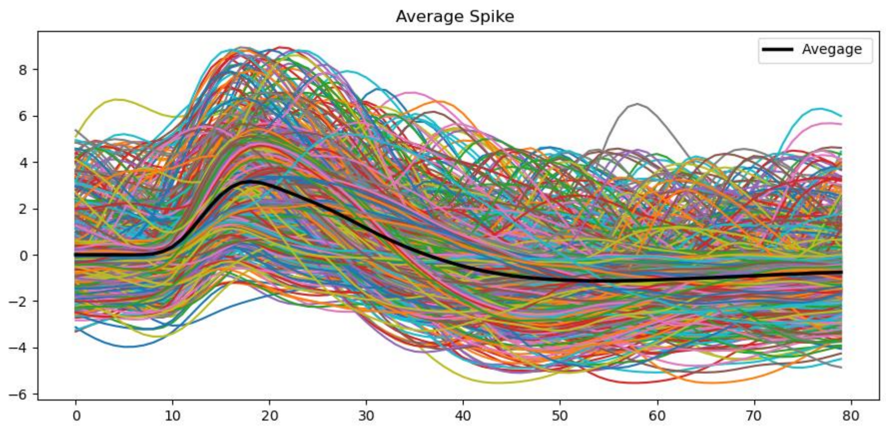


Figure 6: Average peak topology

The length of the average peak is roughly 50 samples. Using this, a final value for the length of data that will be input into the neural net is chosen as 75 to cover for variance in peak length. *Figure 7* shows 6 examples for frames that were created for input into the neural net.

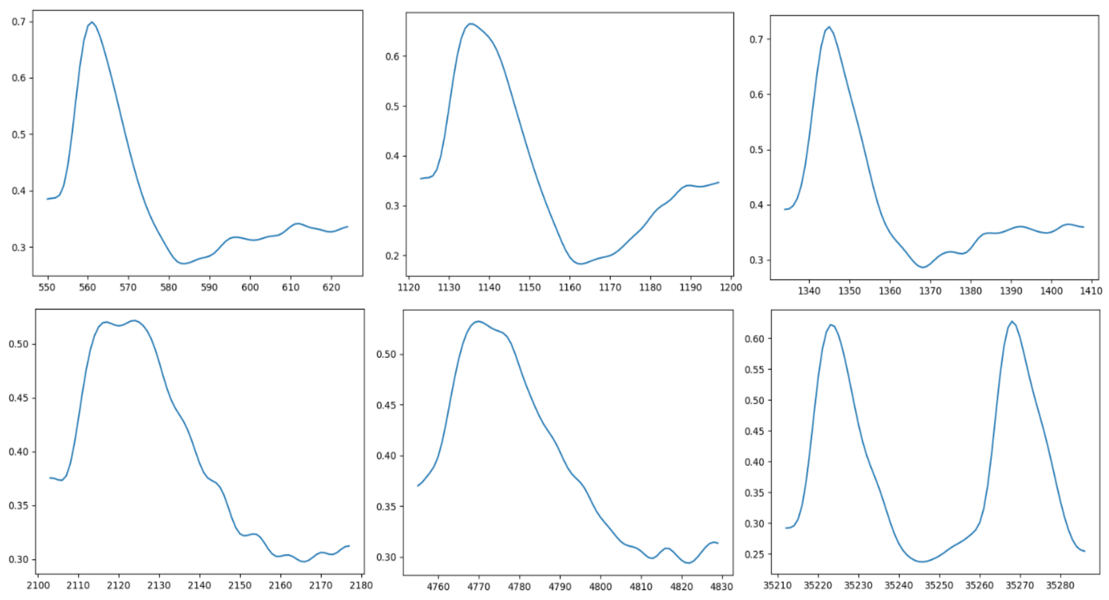


Figure 7: Example windows

3.4 Multilayer Perceptron

Neural networks aim to mimic the development of neurons in the brain. Using layers of these artificial neurons, complicated tasks such as classification can be achieved with high accuracy and consistency. It is for that reason that the multilayer perceptron was chosen for the task of classifying different peaks in a dataset [1].

3.4.1 Topology

To match the data size, there are 75 input nodes, one for each sample. These are then fed into the hidden layer and then passed from the hidden layer to the output layer in a fully connected, feed forward arrangement. The output layer contains 5 nodes, one for each neuron type. Deciding the hidden layer node count is an analytical task that varies from situation to situation. Until further optimisation was carried out, a preliminary skim through node counts showed that a hidden layer count of 125 did not add too much throughput time while offering relatively high levels of accuracy. *Figure 8* Shows the topology of the network.

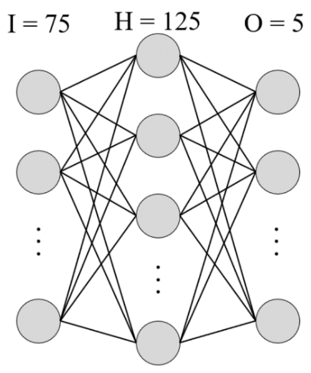


Figure 8: Network topology

3.4.3 Training and Testing

The training data was split into training and test samples, a common breakdown when developing neural networks [2]. The neural network was trained on the 80% data using the predetermined correctly detected peaks. The number of epochs used in training showed large improvement at first but plateaued after only a few iterations. *Figure 9* shows the increase in accuracy with increasing epochs.

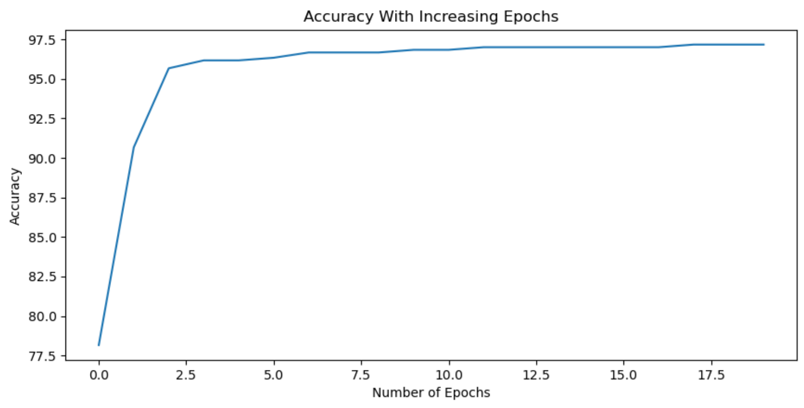


Figure 9: Training repetition analysis

For the final submission, the neural net is trained on the entire training dataset to avoid missing out on valuable training data as none needs to be saved for internal testing.

3.5 Performance

When testing withing the test dataset, the MLP shows good performance, reaching 97% accuracy within very few epochs. The confusion matrix for this test run of 20 epochs is shown below in *Figure 10* alongside a plot of the incorrect guesses for each class in *Figure 11*.

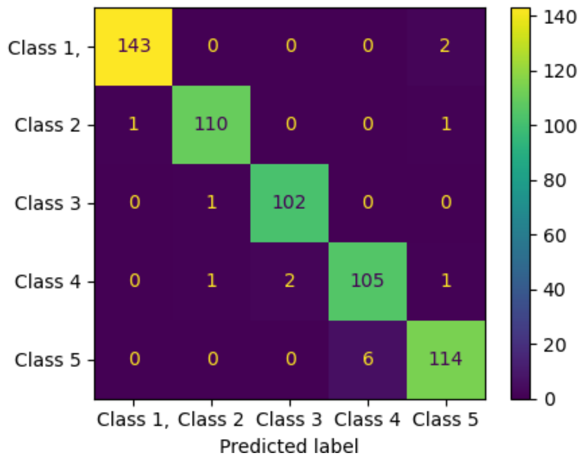


Figure 10: Training result displayed in a confusion matrix

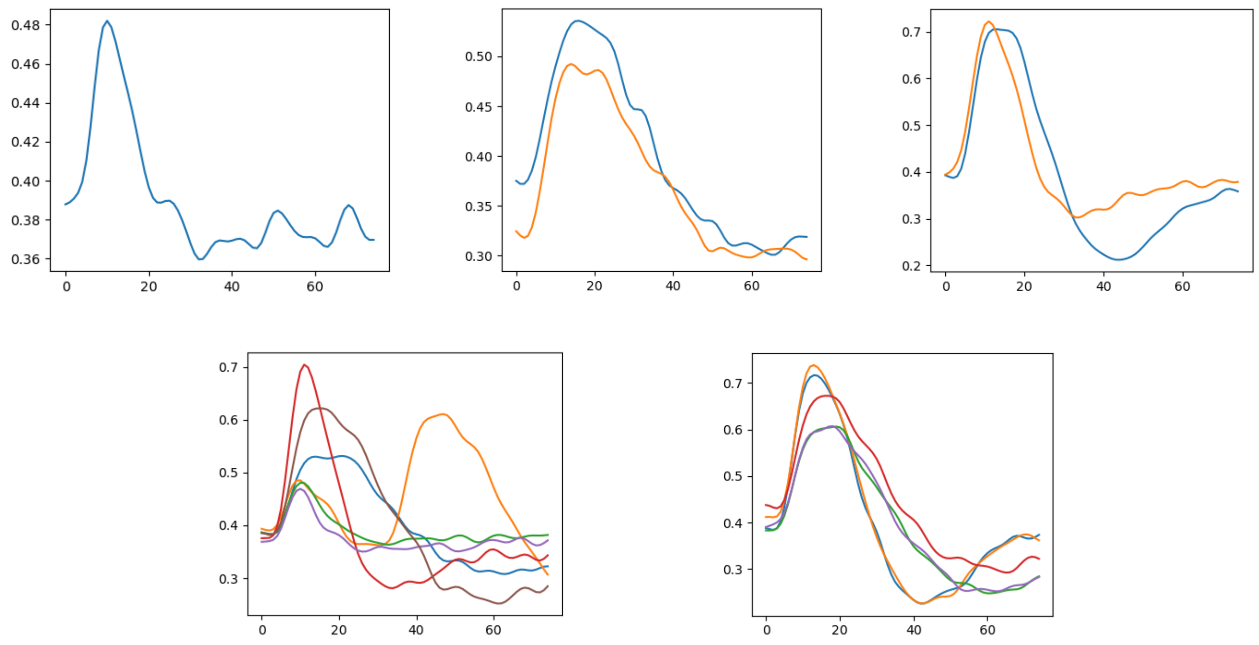


Figure 11: Peaks labelled incorrectly

The incorrectly labelled peaks don't show any extremely obvious trends, making it difficult to deduce where the MLP is falling short and adjust.

When applied to the submission data, the model detected the peaks as shown in . The spread is skewed but not unreasonable indicating that it could potentially have some success despite the added noise and drift of the samples.

Table 2: Detected classes from submission data

Class	Number of Detections

0	497
1	432
2	1151
3	1287
4	64

4. K-nearest neighbour

K-nearest neighbour uses a simpler method of similarity checking to classify the peaks. By first extracting features that vary the most between the classes, the features can be plotted against each other and the distance from a test sample to the nearest 'K' neighbours found. The neighbour's classes can then be summed and used to guess the class of the test data point [4]. A visualisation of how this process works is shown below in *Figure 12*.

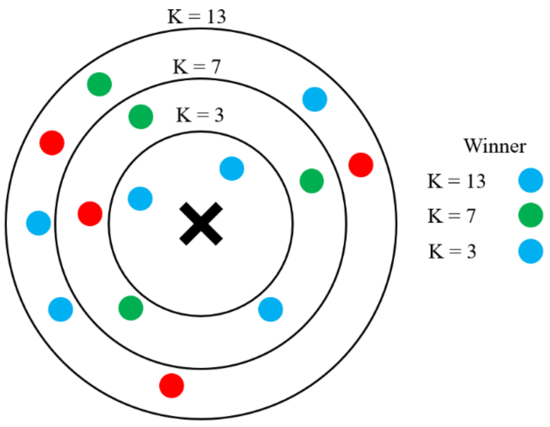
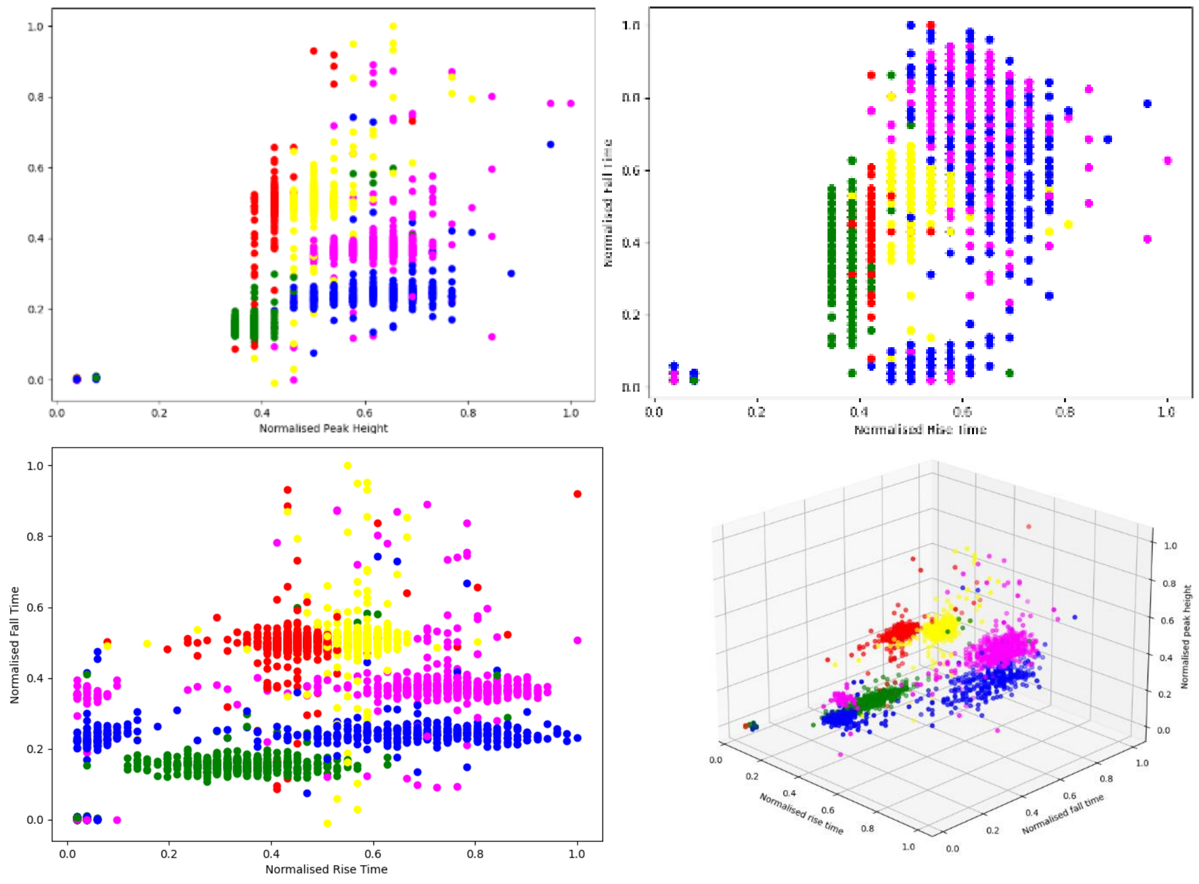


Figure 12: K-nearest neighbour example

4.2 Feature Extraction

The first step after pre-processing is feature extraction. Deciding the features that show the most variance between the neuron types is more likely to lead to a higher accuracy of the K-nearest neighbour algorithm, as there will be less spread of neuron types in each 'neighbourhood'. These features were chosen as the height of the peak, the rise time and the fall time. Extracting the features from the data is a simple process thanks to the amount of pre-processing carried out up to this point. The windows generated based on the peak detection are used alongside calculations of local minima and maxima to generate the three features. Figure x shows the comparison of these features and their relative spreads.



These plots show how adding more features offers more separability, as if only figure B was used, it would be incredibly difficult to differentiate between the blue and magenta datapoints, however by comparing against the other two features, they are much more easily separated.

4.3 Principal Component Analysis

The concept of principal component analysis (PCA) seems very appealing when dealing with K-nearest neighbour classification. The purpose of PCA is to reduce the complexity of the data, through dimension reduction, without losing any useful information. *Figure 13* shows the effect of PCA analysis on the three features reducing them down to two dimensions. The loss of information deemed too costly as the performance benefits were not useful for such a small dataset. The performance of the K-nearest

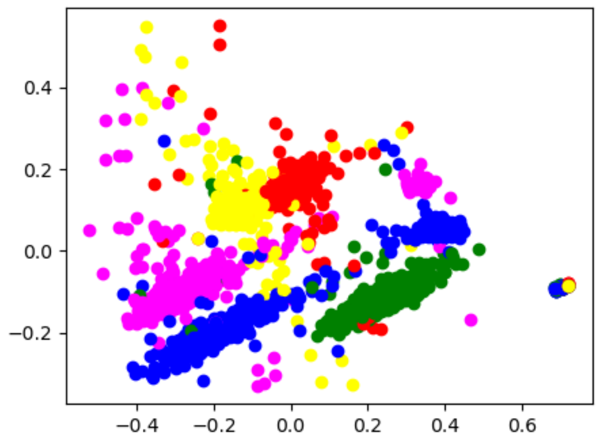


Figure 13: Result of PCA

neighbour algorithm was compared with and without. The performance without was found to be 1.6667% worse.

4.3 Nearest Neighbour

Using the extracted features dataset, the nearest neighbour analysis can be carried out. The training dataset was once again split 80:20. For each data point in the test data set, the Euclidean distance to every point in the training set was calculated and sorted. The K-nearest neighbours were then shortlisted, and their classes analysed, with the most common class ‘winning’ and being chosen as the label. This process was repeated until the entire test dataset had been covered and the

4.4 Performance

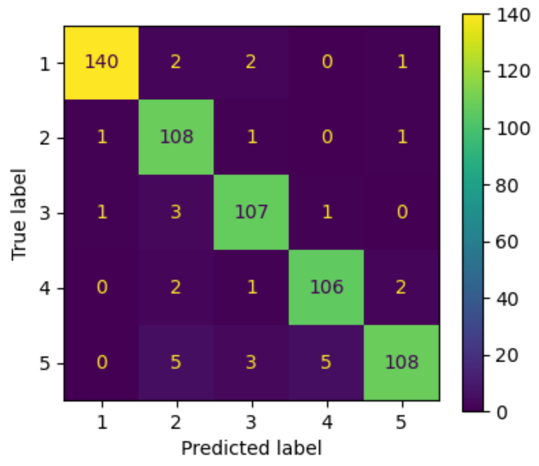


Table 3: Detected classes from submission data

Class	Number of Detections
0	1237
1	461
2	1027
3	442
4	264

5. Optimisation

The optimisation method chosen was a genetic algorithm (GA). The GA was applied to the task of optimising the number of hidden layers as well as the learning rate simultaneously. This meant that each individual contained two genes, one for each feature. The epochs were kept low (e = 3) to avoid excessive computation time and the algorithm was run for 10 generations in order to extract the best set of parameters within a reasonable time frame. *Figure 14* shows how the fitness of the population decreased as the generations increased, showing that adding extra generations would not provide significant further optimisation.

The result of this test is the optimal parameters as follows: learning rate = 0.38, hidden layers = 800. With this setup, the MLP achieved an accuracy of 97.96 % during testing on the correctly detected peaks.

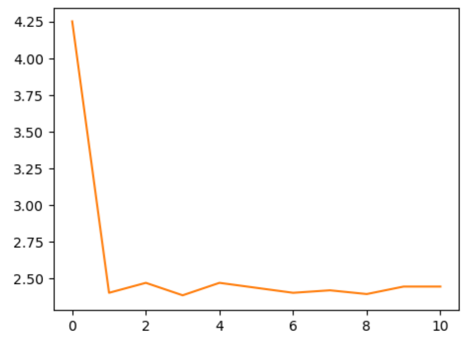


Figure 14: Population fitness with increasing generations

6. Conclusions

Overall, there is some good evidence of a strong performing MLP and K-nearest neighbour analysis for the classification of spikes during the training and testing of the training data. It can easily be argued however that this could not correlate to the performance in the submission dataset. Figure 15 shows an example of how the peak detection function struggles with the extra noise leading to far more false positives. This would require either further pre-processing or a slightly altered approach to avoid. The issue becomes that if the design is tailored to match the training data, it may not succeed for the submission data due to the excess noise and different peak shape, but if processing focuses on the best performance on the submission data, it will not perform as well on the training data, affecting training performance and efficacy. In an ideal world there would be two signal processing schemes that normalise the datasets to be as similar as possible, providing both the MLP and the KNN methods a better chance of success.

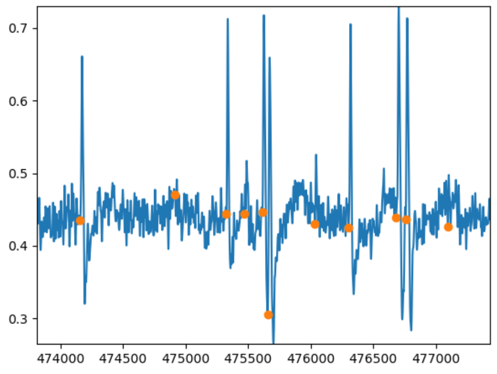


Figure 15: Submission dataset peak detection

7.Bibliography

[1] O. Knocklein, "Classification Using Neural Networks," [Online]. Available: https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f. [Accessed 16 December 2021].

[2] J. Brownlee, "Train-Test Split for Evaluating Machine Learning Algorithms," Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/. [Accessed 16 December 2021].