**METHODOLOGY**

**Generating signals**

The signals used in this study are EEG signals which were generated by using Python 3.8. Two types of signals were generated, healthy EEG signals and degraded EEG signals. At first, eight healthy and eight degraded EEG signals were generated via the NumPy toolbox using Python. Since EEG waves are emitted from the brain, they are sinusoidal in nature and superimposed with different noises which can be due to the signal disturbances from the rest of the biological systems (such as the heart), the impedance of the signal acquisition system, and background signals. For this purpose, random noise was added to the signals to make them as realistic as possible. For this study, only a single-channel EEG signal was sufficient, with a sampling frequency of 1000 Hz. Each signal was 10 seconds long.

For generating the degraded signals, all parameters were kept the same. Additionally, to ‘degrade’ the signal, Gaussian noise was added. For 4 signals, the amplitude for Gaussian noise was kept at 0.5 whereas for the remaining 4, it was kept at 0.9.

Later on, during the first trial of taking the signals as input in the Machine Learning models, overfitting was noticed so the number of signals was increased to eighteen for each type.

**Code for generating signals:**

import numpy as np

import matplotlib.pyplot as plt

def generate\_dummy\_eeg(duration=10, sampling\_rate=1000, noise\_level=0.5):

time\_points = np.arange(0, duration, 1/sampling\_rate)

# Generate random noise

noise = np.random.normal(0, noise\_level, len(time\_points))

# Create a sine wave resembling brainwave patterns

theta = 8 # Frequency in Hz

sine\_wave = np.sin(2 \* np.pi \* theta \* time\_points)

# Combine noise and sine wave to simulate EEG signal

eeg\_signal = noise + sine\_wave

return time\_points, eeg\_signal

duration = 10 # seconds

sampling\_rate = 1000 # Hz

noise\_level = 0.5

time\_points1, eeg\_signal1 = generate\_dummy\_eeg(duration, sampling\_rate, noise\_level) #generation of a single signal

**Code for the addition of Gaussian noise to generate degraded signals:**

noise\_amplitude = 0.9 # Adjust the noise amplitude as needed

gaussian\_noise = np.random.normal(0, noise\_amplitude, len(eeg\_signal1))

eeg\_with\_noise1 = eeg\_signal1 + gaussian\_noise #add noise to the generated signal

It is important to note that the same healthy signals were not degraded, but new signals were created for the purpose of degrading them to remove any biased readings.

**Feature extraction**

EEG features become more robust in the frequency domain. More importantly, when dynamic signals are decomposed into power spectrums, they become more easily understandable, especially for machine learning purposes. For this reason, the Welch periodogram was used to compute the power spectral densities of the data. The SciPy toolbox of Python was incorporated. The parameters of the welch function were:

A 4-sec window length

50% overlapping

A ‘Hanning’ window type

These parameters were set for both, healthy and degraded signals. As a result, each 10-second signal produced 2001 power spectral densities.

Code for applying Welch periodogram:

import scipy

from scipy import signal

import numpy as np

import pandas as pd

healthy\_data1 = pd.read\_csv(r'C:/Users/PC/Desktop/sandra/data\_healthy/eegh9.csv', header=None)

healthy\_data1 = np.transpose(healthy\_data1)

raw\_data1 = np.array((healthy\_data1))

fs = 1000

wl = fs\*4

pwelch1 = signal.welch(raw\_data1,fs=1000,window='hann',nperseg=wl,noverlap=wl\*0.5,nfft=wl)

hsig1 = pwelch1[1]

np.savetxt('healthy\_w9.csv', hsig1, delimiter=',')

**Data analysis**

To check whether or not the data of both types was drastically different, the mean power spectral densities of 8 signals for healthy and degraded were calculated respectively. Since the signals were generated using a normal distribution, parametric tests could be performed on them. To compare the healthy signals with degraded ones, t-test was applied. This t-test was two-sample, assuming unequal variances, and two-tailed. The results show that there is no significant difference between the power spectral densities of the healthy signals and the degraded signals (p > 0.05) so it can be said that it is very difficult to differentiate between the two types of signals using normal analyses, so the idea of using machine learning is a necessary intervention.

|  |  |  |
| --- | --- | --- |
|  | *healthy* | *degraded* |
| Mean | 0.001486 | 0.002507 |
| Variance | 0.000983 | 0.00095 |
| Observations | 2001 | 2001 |
| Hypothesized Mean Difference | 0 |  |
| df | 3999 |  |
| t Stat | -1.03899 |  |
| P(T<=t) two-tail | 0.298871 |  |

Table 1: t-Test between the mean healthy and mean degraded signals: Two-Sample Assuming Unequal Variances

**Data Arrangement**

Data was merged into a single array of 36 x 2002, where the rows denoted the number of signals and columns (1-2001) denoted the Power Spectral Densities. A single column of ‘labels’ was added to the data. This column had two labels, ‘healthy’ and ‘degraded’. The merged data was shuffled and divided into 80% training and 20% testing data.

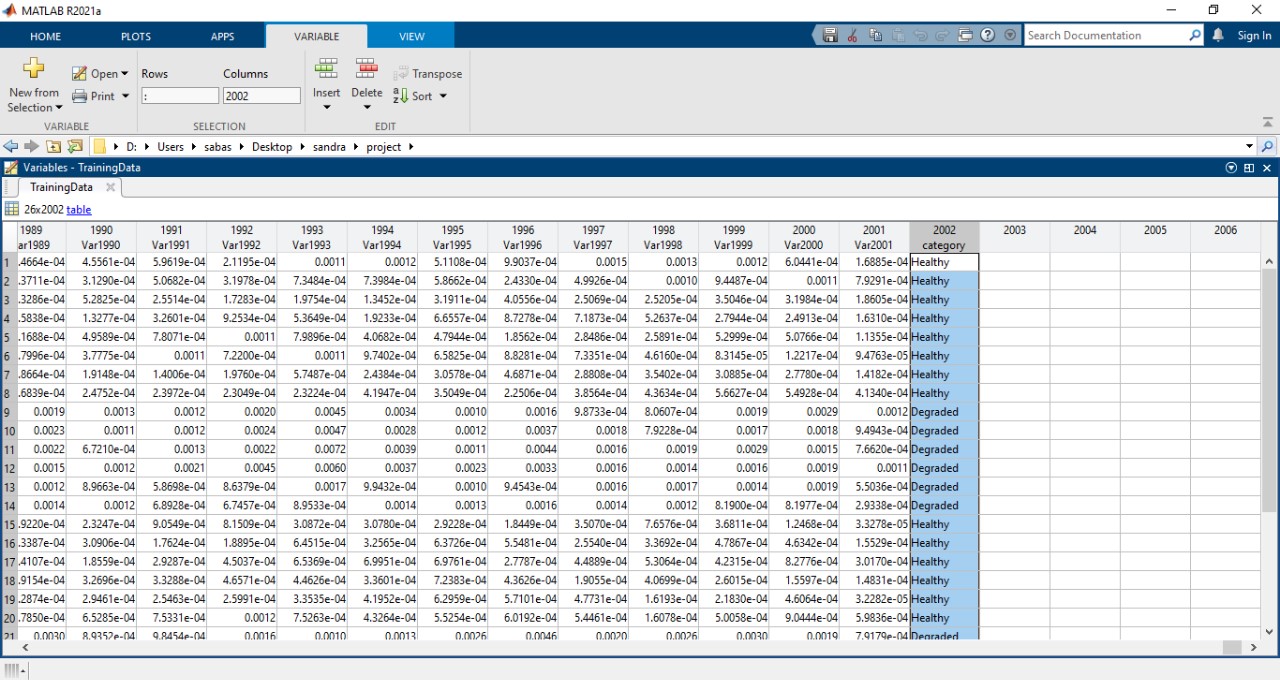


Figure 1: Data arrangement of training data

**Building the Model**

All classification tools and machine learning algorithms were developed using MATLAB R2021a (The MathWorks Inc., Natick, 2021). Four built-in classifiers were used within the Classifier Learner app on MATLAB, namely, Support Vector Machine (SVM), Ensemble, K Nearest Neighbour (KNN), and Artificial Neural Network (ANN). These classifiers were chosen after surveying various research papers based on the classification of EEG signals.

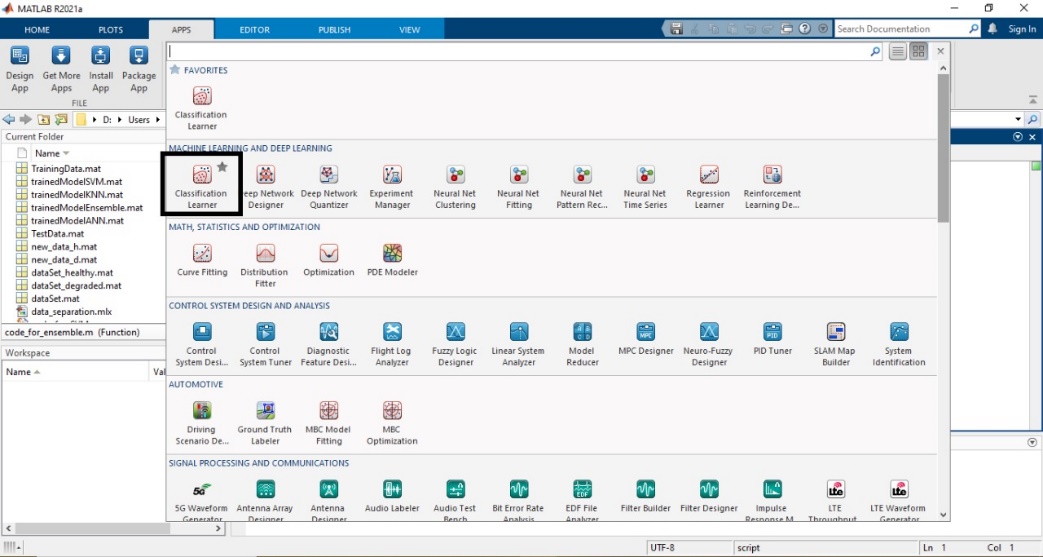


Figure 2: Classification Learner application module in MATLAB

The validation method used was the K-fold method with 5 folds and 2 iterations. The k-fold method (in this case 5-fold) is a cross-validation method which makes five subsets of the dataset and uses four subsets for training and one for validation. This method is repeated five times until all the subsets have been validated. The k-fold validation method is used to remove over-fitting. Overfitting is a common problem in machine learning where a model learns to capture the noise or random fluctuations in the training data rather than the underlying pattern or relationship. One of the challenges that are faced when applying ML to EEG data is the higher chances of overfitting, therefore, cross-validation was performed.

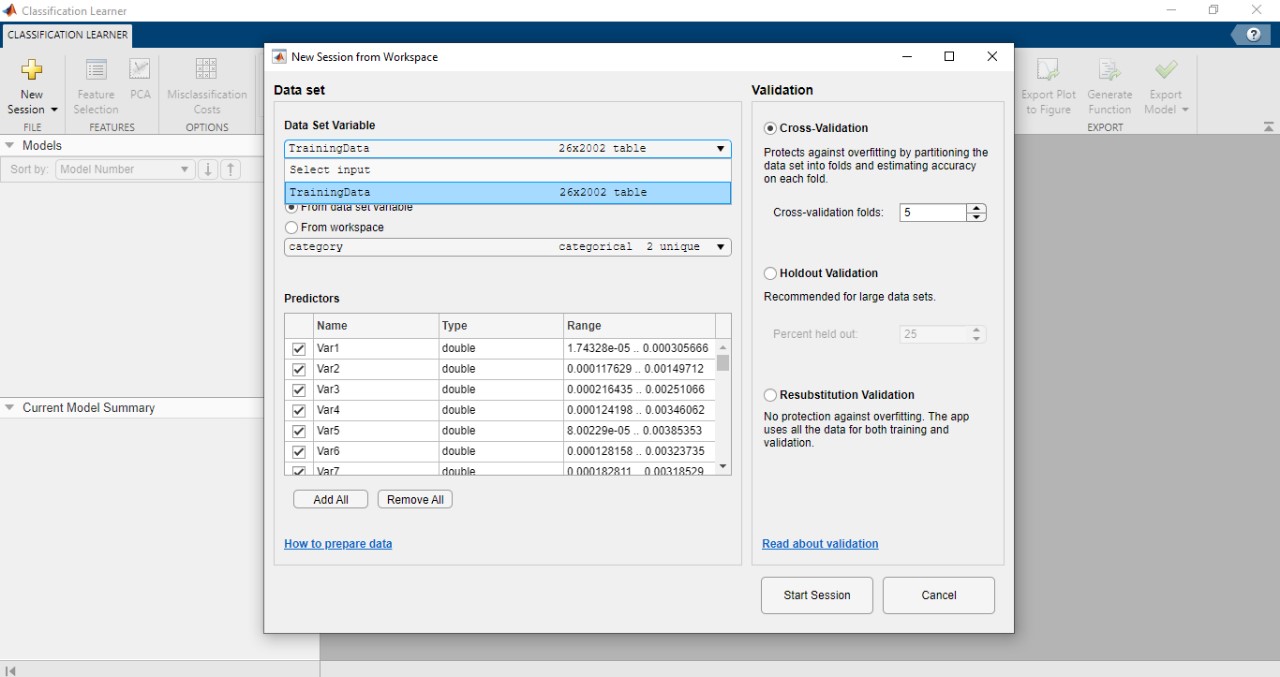


Figure 3: Data entry and validation

**Parameters of models**

*Artificial neural network (ANN)*

Artificial neural networks are versatile and powerful tools in machine learning, capable of learning complex patterns and relationships in data and making accurate predictions across a wide range of applications. ANNs are used widely in classifying EEG signals, especially in the frequency domain or in the form of power spectral density. The activation function used was ReLU. An activation function introduces nonlinearity into the network, allowing it to learn and model complex relationships in the data. ReLU, which stands for Rectified Linear Unit, is one of the most commonly used activation functions in ANN due to its simplicity. One fully connected layer was used with five neurons.

*Support Vector Machine (SVM)*

SVMs are a supervised learning algorithm used for classification and regression tasks, although they’re primarily known for their effectiveness in classification. SVMs are popular for EEG signal classification due to their ability to handle high-dimensional data and their robustness against overfitting. Two iterations were used; the kernel function was set to linear. Bayesian optimizer was used and the multiclass method applied was one vs one.

*K nearest neighbour (KNN)*

KNN is considered a non-parametric, lazy learning algorithm, meaning that it doesn't make any assumptions about the underlying data distribution and doesn't learn a specific model during training. Instead, it memorizes the entire training dataset and makes predictions based on the similarity between new data points and the existing training examples. KNN is a simple and intuitive algorithm that is used in EEG classification for its ease of implementation and effectiveness, especially in scenarios where the underlying data distribution is not well understood. The hyper-parameter values were set to four neighbours, Minkowski (cubic) distance metric, and inverse distance weight.

*Ensemble*

An ensemble is a technique that combines the predictions of multiple individual models (often called base learners or weak learners) to produce a more accurate and robust prediction. The idea behind ensemble learning is that by combining the predictions of multiple models, the weaknesses of individual models can be mitigated, and overall performance can be improved. Ensemble learning techniques can be effectively applied to EEG (electroencephalography) classification tasks to improve prediction accuracy and robustness. The ensemble method used here was LogitBoost. This method specifically targets improving the performance of binary classification tasks by directly modeling the class probabilities.

**RESULTS**

Table 2: Validation and testing accuracies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accuracy | ANN | SVM | KNN | Ensemble |
| Validation | 96.2% | 100% | 80.8% | 88.5% |
| Testing | 100% | 100% | 80% | 90% |

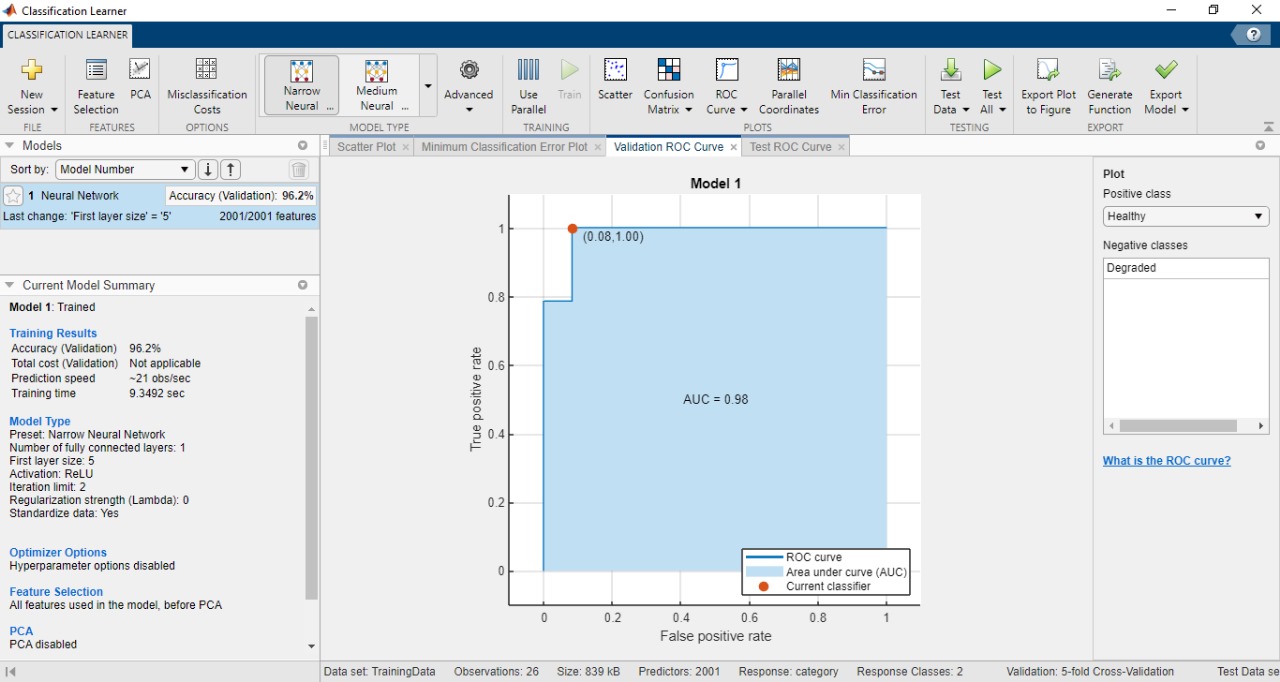


Figure 4: ANN validation details and ROC curve

ROC curve stands for Receiver Operating Characteristic curve. It's a graphical representation commonly used to evaluate the performance of binary classification algorithms. The ROC curve visualizes the trade-off between the true positive rate (sensitivity) and the false positive rate (1 - specificity) across different decision thresholds.

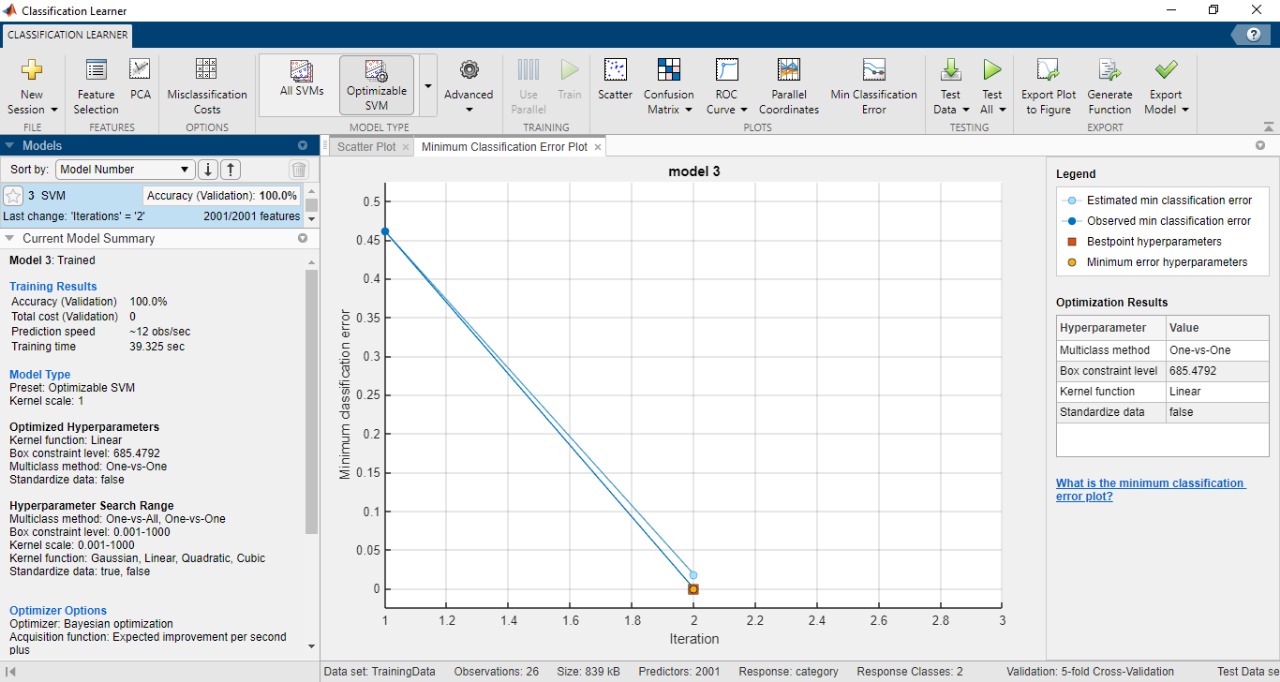


Figure 5: SVM validation details and classification error plot

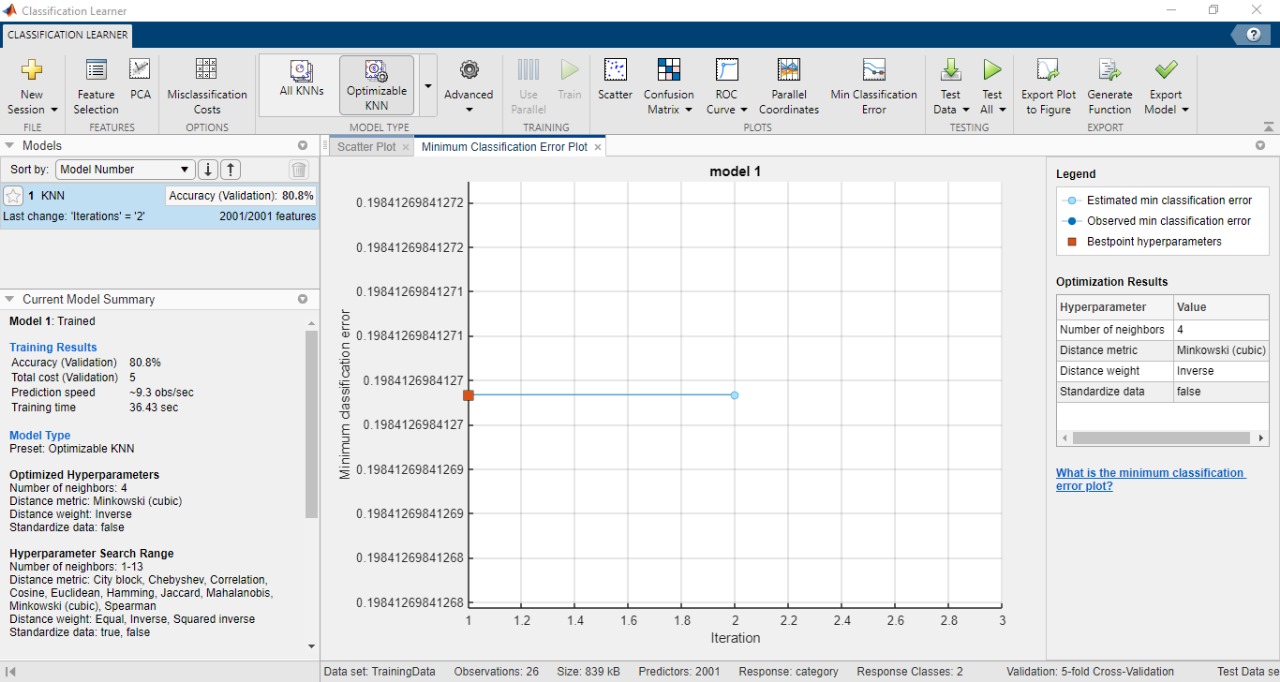


Figure 6: KNN validation details and classification error plot

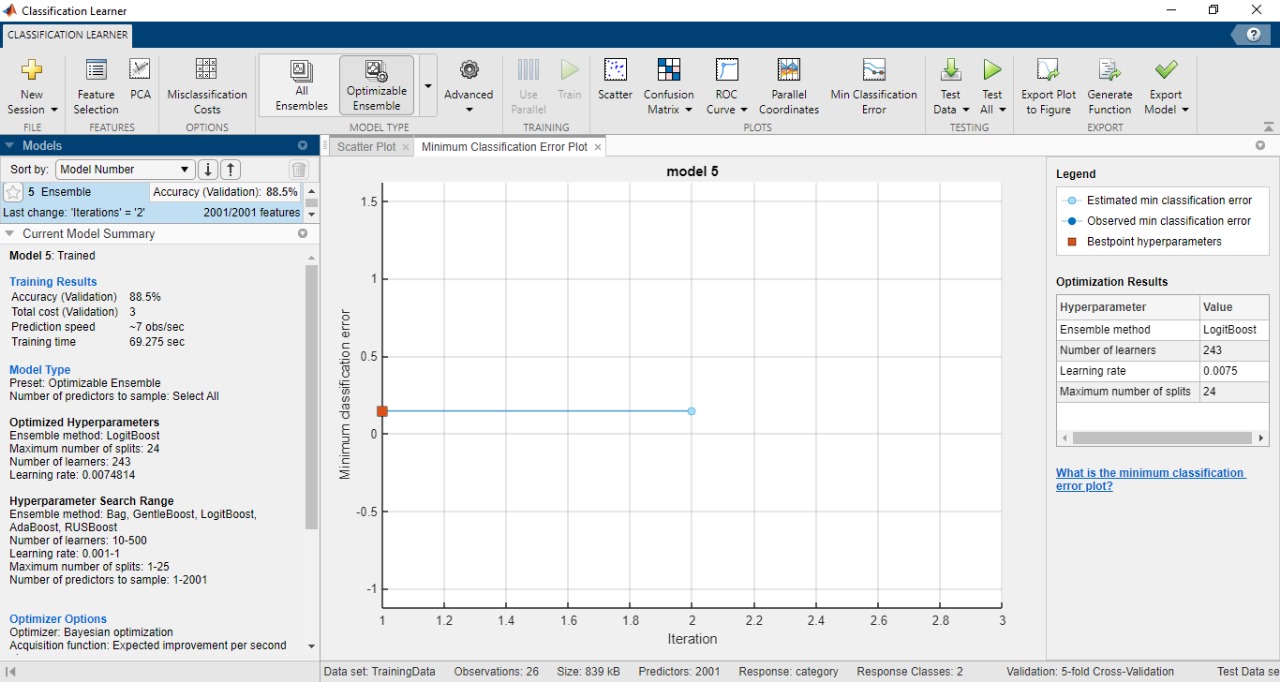


Figure 7: Ensemble validation details and classification error plot

**Confusion matrices**

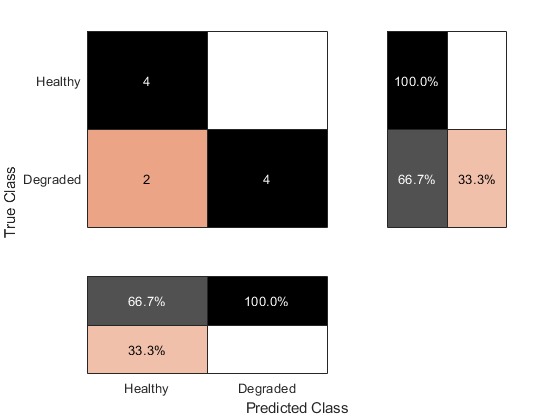
****

Figure 8: KNN confusion matrix

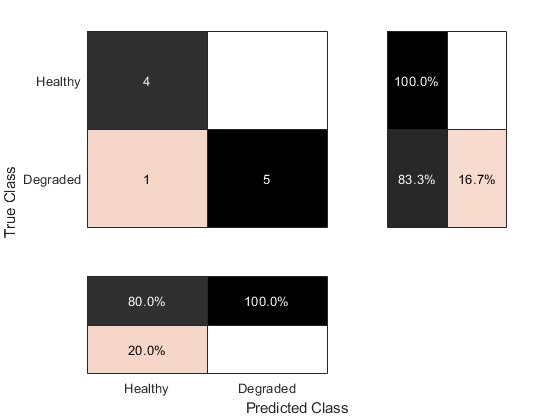
****

Figure 9: Ensemble confusion matrix

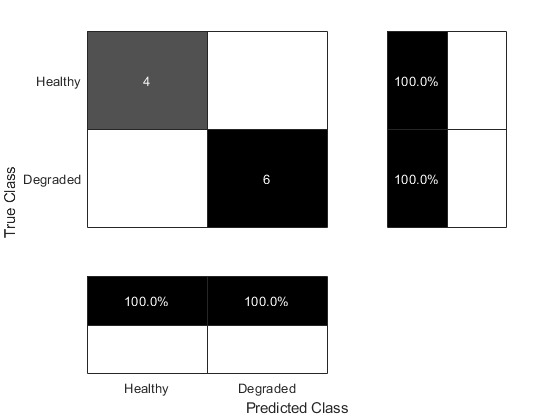
****

Figure 10: SVM confusion matrix

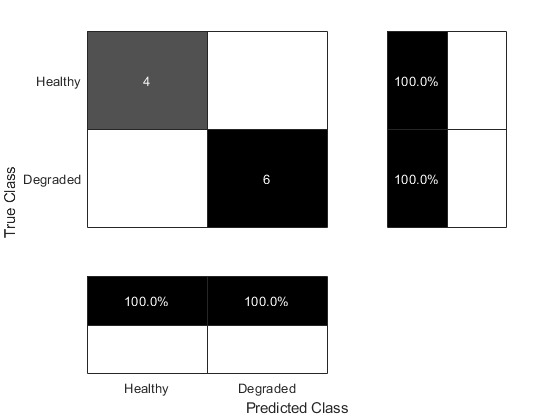
****

Figure 11: ANN confusion matrix