5-2019

A Machine Learning Technology for Rapid Detection of Carbon Nanotubes/DNA Hybridization in Biosensor Healthcare Applications

Steven K. Ang
University of New Haven

THE UNIVERSITY OF NEW HAVEN

A MACHINE LEARNING TECHNOLOGY FOR RAPID DETECTION OF CARBON NANOTUBES/DNA HYBRIDIZATION IN BIOSENSOR HEALTHCARE APPLICATIONS

A THESIS

submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE in COMPUTER SCIENCE

BY

Steven K. Ang

University of New Haven
West Haven, Connecticut
May, 2019

ACKNOWLEDGEMENT

I would first like to thank my thesis advisor, Dr. Said M. Mikki, of the Tagliatela College of Engineering, University of New Haven, for all his help in bringing this study together. Whenever I encountered any trouble or had questions about the study, his door was open. He gave me the freedom to take my research in new directions, while offering the objective and constructive feedback necessary for me to keep the study focused and on-track.

I am grateful to the former Dean of the College of Arts and Sciences, Dr. Alverez Lourdes, for giving me the opportunity to study here at the University of New Haven. I could not be here without her kindness and consideration. My thanks also go to the ALLEX Foundation for providing me with this great opportunity through their fellowship program.

Finally, I wish to thank my dear wife, Yi-Heng, and our precious baby, Orville Reeve Ang. Thank you for the patience and support you have shown me throughout this journey.

ABSTRACT

In molecular biology, the term "DNA hybridization" generally refers to the process of forming a double stranded nucleic acid from joining two complementary strands of DNA. The degree of genetic similarity of the DNA resulting from hybridization can be detected either by using the chemical characteristics of DNA samples or by utilizing reliable biosensors which transform the chemical characteristics into a source of electrical measurements. In past research about such sensors, known as DNA Hybridization Detection Systems, the thermal and electrical characteristics of carbon nanotubes are utilized to detect whether hybridization takes place or not. However, human interpretation of the measured data can lead to uncertainty regarding, which compromises one crucial characteristic of biosensors—reliability.

Research aimed at greater understanding of this sensor is still very much underway. This study is intended to make a significant contribution to the growing field of biotechnology by means of analyses of machine learning methods from a classification perspective. The ultimate goal of this thesis is to see if any of the existing classification algorithms, such as k-nearest neighbors and decision trees, are capable of predicting the state of hybridization based on simple electrical measurements. In this way, the machine learning algorithm uses real-life data to provide a systematic tool that can be utilized in various fields, such as bioinformatics, biomedicine, and forensic science.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

DNA, also known as deoxyribonucleic acid, is a molecule that contains the genetic information of every living organism. During cell reproduction, DNA plays the role of passing down this genetic information from parents to their offspring [1]. In molecular biology, DNA hybridization generally refers to a technique that measures the degree of genetic similarity of DNA. Nowadays, measurements of this genetic similarity usually make use of biosensors. In past research about these sensors, DNA Hybridization Detection Systems[2] utilizing excellent thermal and electrical characteristics of carbon nanotubes were employed to detect whether hybridization of DNA takes place or not.

Carbon nanotubes are nano-scale cylindrical structures composed of one layer of carbon atoms. They typically exhibit unusual electrical, mechanical, and thermal properties and have been applied in a variety of ways. By combining carbon nanotubes with DNA, it is possible to investigate hybridization of specially prepared solutions containing both DNA samples and nanotubes. The interaction between carbon nanotubes and unknown DNA strands can be put into use to devise a new generation of biological sensors exploiting the potential of promising nanomaterials such as carbon nanotubes.[3]

In order to detect hybridization, biosensors are usually designed and deployed to measure the electrical characteristics of solutions containing mixtures of carbon nanotubes and DNA. This is done by measuring the electrical conductivity of these mixtures. Consequently, the

device can provide valuable information about the state of the mixture after which the goal is to determine whether hybridization took place or not. These special devices are known as nanobiosensors, and they form the backdrop of this study.

Biosensors have huge potential in many different fields, including biomedicine, forensic science, engineering, and others [4] [5]. One crucial characteristic of biosensors is their reliability. Research aimed at understanding the sensor is still ongoing, with the long-term goal of guaranteeing that nano-based biosensors can measure, record data and make an inference with new data accurately. Such ambitions, however, have to contend with the fact that the interpretation of the measured data still relies on human beings with highly specialized knowledge. Such interpretation and limitations in processing results can cause uncertainty with respect to the overall accuracy of the results. One solution to this kind of problem is utilizing machine learning.

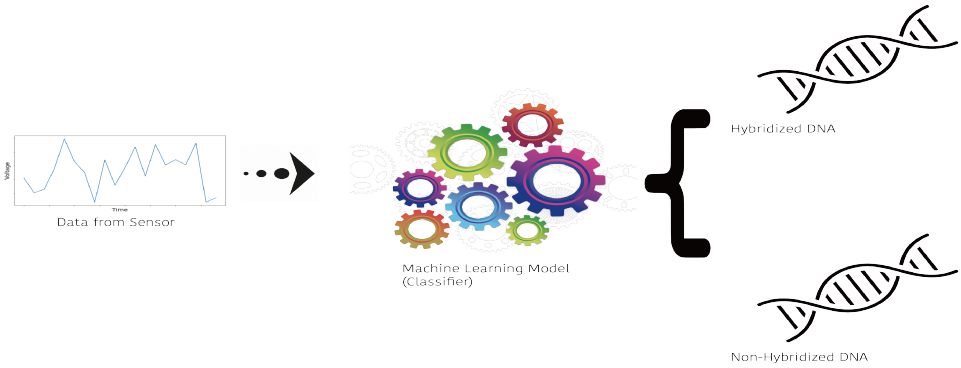

Figure 1.1: The measured electronic voltages will be used as an input of the machine learning algorithm and used to predict whether hybridization took place or not.

Machine Learning is a field in computer science in which computer scientists attempt to make computers learn and think like human beings. In machine learning, available data and a selected algorithm are used to produce and train a model. This model can be used for making inferences in the future when new data becomes available. The inferences will be made using the new data based on the rules the model learned during the training phase. In this way, the uncertainty which is always involved in interpretation of data by human beings can be eliminated. [6]

This study offers a contribution to the new and growing field of biotechnology through machine learning methods from a classification perspective. The ultimate goal of this study is to see if any of the existing machine learning algorithms, such as k-nearest neighbors, decision trees, and other algorithms, are capable of predicting the state of hybridization based on simple electrical measurements (see Figure 1.1). As will be shown in some detail, machine learning algorithms use real-life data to provide a systematic tool that can be part of other related, highly practical applications.

Chapter 2

Biosensor

The biosensor developed by Dr. Sinha and his team is comprised of two major parts. The first is the biosensor unit itself, which has four chips and a small single-board computer, not unlike a Raspberry Pi or Arduino. This device aims to utilizes the excellent thermal and electrical characteristics of carbon nanotubes to detect whether DNA hybridization takes place or not. The second component of the biosensor is an as yet undisclosed material used by Dr. Sinha. Carbon nanotubes will be applied to this material and then it will be laid over the chips.

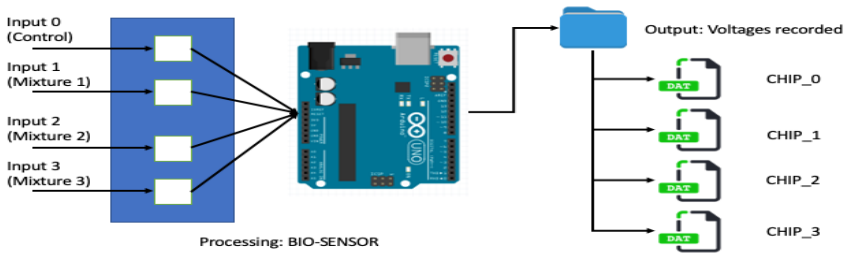

Figure 2.1: The biosensor developed by Dr. Sinha, which is used for this study.

The output of the biosensor used for this study is a folder containing four files. Each file corresponds to a chip of the biosensor:

- CHIP_0 contains the voltage recorded in Input 0

- CHIP_1 contains the voltage recorded in Input 1
- CHIP_2 contains the voltage recorded in Input 2
- CHIP_3 contains the voltage recorded in Input 3

These are the data to be used to determined whether the hybridization took place or not

Three mixtures were prepared for the experiment. The first was the control, which is a TE Buffer with a pH value of 8.0 and without DNA. The second mixture contained a single-stranded DNA, which is the target to be tested. Then the third mixture contained another single-stranded DNA. Dr. Sinha's team are testing whether this third mixture will be hybridized with the single-stranded DNA in the second solution or not. What follows is a brief description of the experiment.

First, the biosensor will be turned on and heated up to a certain temperature (as yet undisclosed). Then, a special device called an "8-Channel Pipette" (see figure 2.2) will be used to drop the prepared mixtures on to the chips. This device can ensure that the user drops the same amount of mixture onto each chip to maintain the integrity of the whole experiment. Each The first drop of each mixture will be injected over the chip when the device reaches the target temperature. The first drop of the mixture must dry up before the second drop of mixture, which contains another single-stranded DNA, can be applied. In the whole experiment, only one drop of each mixture is needed.

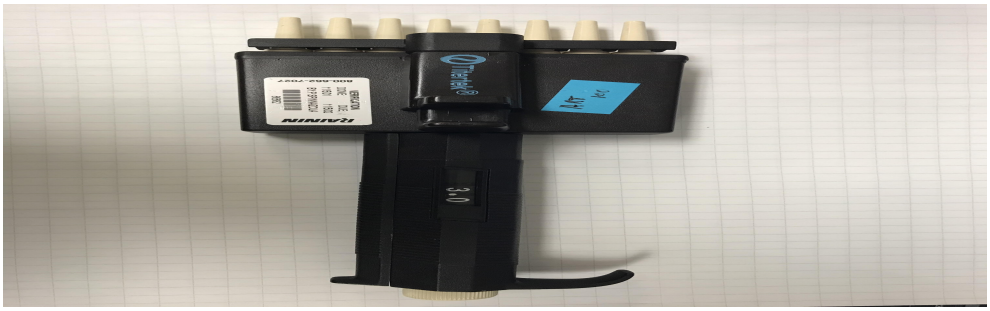

Figure 2.2: 8-Channel Pipette

The second mixture contains an unknown single-stranded DNA, the identity of which we want to ascertain. The third mixture can contain any single-stranded DNA. For the purposes

of a description of the experiment, let us assume the third mixture contains a single-stranded DNA of a virus. If the single-stranded DNA of the second mixture hybridized with the single-stranded DNA of a virus in the third mixture, then we can conclude that the unknown DNA is, in fact, that specific virus.

The biosensor aims to measure the electricity detected and record it accordingly. The recorded voltage, hereafter referred to as the "signal", is used by the team for analysis and determining whether hybridization takes place or not. One problem which this new technology aims to resolve is that present means of analysis rely upon the naked eye and highly specialized or domain-specific knowledge. Using the recorded signal, the team drew a line graph, formed to determine whether hybridization took place or not. Figure 2.3 is a sample of a graph with the data representing whether hybridization takes place, and figure 2.4 represents the results of an experiment in which hybridization did not occur.

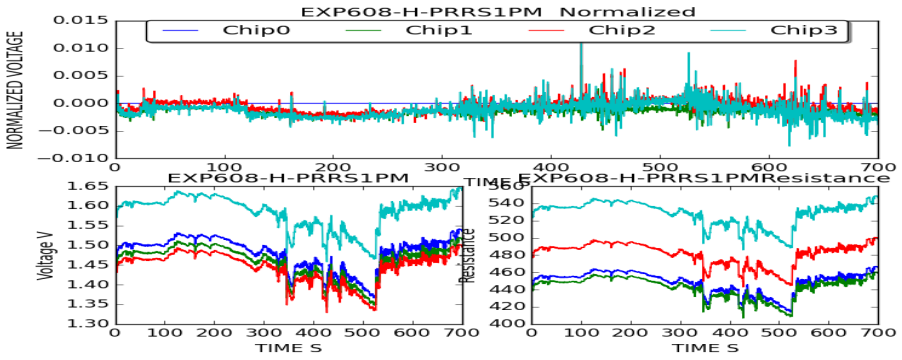

Figure 2.3: Graph of the data pertaining to when hybridization took place.

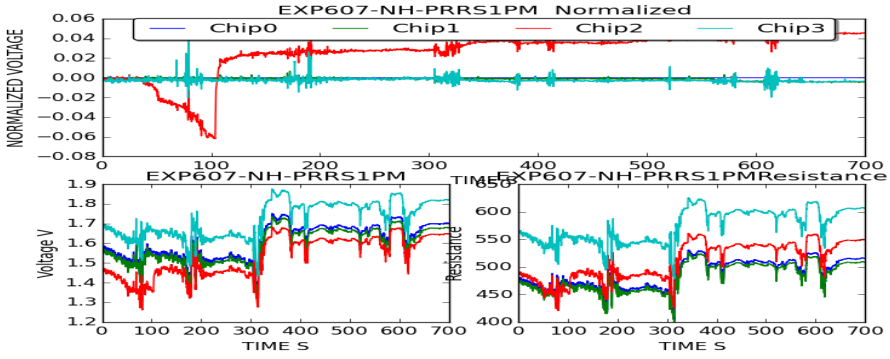

Figure 2.4: Graph of the data pertaining to when hybridization did not occur. .

For someone to be able to make high accuracy classifications and determine whether hybridization takes place or not, a long period of training and an accumulation of practical experiences are both necessary. Such conditions for accurate analysis to be at all possible imply that those with the competence to perform analyses will be small in number, and will still face the methodological problems mentioned above. For most cases, Dr. Sinha and his team assume that users have little or no experience of analysing and classifying hybridization using this technology. Dr. Sinha and his colleagues therefore need to make this device usable for people who may not have any relevant knowledge about it. Machine learning can provide a solution for this problem through the building of models which can replace human beings in making inferences and predictions. To reiterate, this is done by using the data provided to determine whether hybridization takes place. This is why machine learning will be studied together with this biosensor in the present study.

Chapter 3

Machine Learning

Machine learning is a field of computer science in which the computer scientist seeks to enable computers to learn and think like human beings. As the computer scientist feeds data and other relevant information in the form of real-world observations and interactions to the computer, the computer can, over time, improve the way it learns and processes information in autonomous ways.[6] One example of real-world information fed to computers in machine learning is the inputting of a vast number of images of human handwriting for the purpose of building a model that can recognize certain handwritten words or phrases. It must be noted, however, that machine learning is limited in the kind of problem it can serve to resolve. When we have problems such as house price prediction problems, with large quantities of data capable of being interpreted by those with relevant knowledge but which cannot be solved by using simple conditional phrases like IF-ELSE statements or by using simple mathematical formulas, then machine learning may be useful.

An example of the kind of problem appropriate for Machine Learning is the famous Titanic Survival Prediction Problem. The data of the passengers, such as their age, gender, and cabin class, as well as their status (either survived or deceased), are inputted into the model to be used. We are certain that there be a pattern in the data, such as the survival rate of males being higher than that of the females, or that, on average, females travelling with children had a higher survival rate than others. We cannot, however, devise a mathematical

formula or a single logic by which we can easily compute whether a certain passenger survived or not. Consequently, this kind of problem is suitable for machine learning.

The data of the passengers, for example, their respective sex, ticket class, age, number of siblings, etc., are what we call "features" in machine learning. Such features serve as an input for machine learning models. In this case, the ultimate goal of machine learning is to discover a function, called f (x), which can generate an output y as a prediction of the results(s) based on a given feature. Based on this description, the following formula can now be presented:

$$\begin{aligned} \mathrm{f(x)} = \ & \mathrm{f(x_1, x_2, ..., x_n)} \\ & = \begin{cases} +1, \sum_{n=1}^{n} \mathrm{w_i x_i} + \mathrm{b} > 0 \\ \text{-1}, \sum_{n=1}^{n} \mathrm{w_i x_i} + \mathrm{b} < 0 \end{cases} \\ & = \mathrm{sign}\left\{ \sum_{\mathrm{n=1}}^{\mathrm{n}} \mathrm{w_i x_i} + \mathrm{b} \right\} \end{aligned} \tag{3.1}$$

Here, w_n stands for the coefficient or weight for each element and b represents a noise. If the sum of all the items is greater than 0, then this function will be returned as +1 to indicate that passenger has survived. The value of -1 represents that the passenger is deceased. The formula shown in equation 3.1 is a sample of a machine learning model which has a linear relationship. However, not every problem can be resolved with linear functions. Some of the problems use non-linear functions, as shown in figure 3.1.

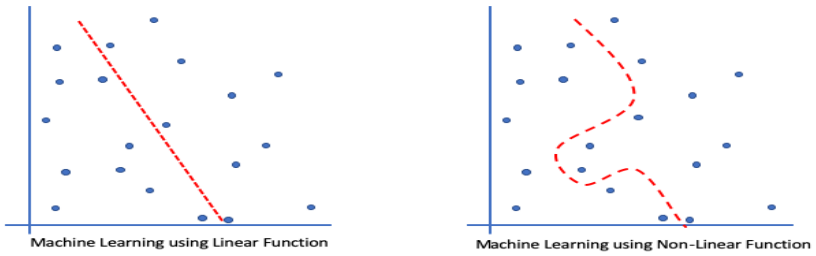

Figure 3.1: Different kinds of Machine Learning

Following the example of a suitable problem given above, our data is presented in table 3.1. There, all items from Passenger ID to Embarked are what we call features, and the item survived is known as the label for this data. Each item in Table 3.1 will become an element of x, such as x1 = PassengerID, x2 = PClass, x10 = Embarked, etc., and an item survived will become our value for y, or y = 1. A single data item is not enough to train a model, so we will have a large quantity of data like this and utilize it to train a machine learning model. The ultimate goal is to train a model which is capable of making predictions with unknown data which has similar structures to the data of the training dataset. We are asking machine learning to find a function such as

$$f(x) = sign\left\{ \sum_{n=1}^{n} w_i x_i + b \right\} \approx y \tag{3.2}$$

where y can be +1, which indicates that a passenger has survived, or -1, which indicates that the passenger was deceased.

Passenger ID	1
PClass	3
Name	Braund, Mr. Owen Harris
Sex	Male
SibSp	1
Parch	0
Ticket	A/5 21171
Fare	7.25
Cabin	
Embarked	S
Survived	1

Table 3.1: Sample data from the "Titanic: Machine Learning from Disaster" prediction problem (taken from the machine learning community website "Kaggle")

It needs to be said that machine learning, as a computer program, cannot directly understand the given data. In other words, as a computer program, it cannot understand the meaning of male. In addition, there might be some data which has missing a missing value or values, such as "Cabin" in table 3.1. Consequently, we need to process and clean the data up before employing it in machine learning. This is what we call "feature engineering",

which will be discussed in the next chapter.

The data used in this study is much simpler than the sample above. It only contains the recorded voltage and timestamp. It is assumed that all the data was recorded accurately by the biosensor and, therefore, that there are no missing values.

3.0.1 Supervised Learning

As the name "supervised learning" indicates, there will be someone to teach the machine learning model the data and tell the model the value and relevance of each data item. In real life, this is done by providing a dataset composed with an input-output pair (the input being the data and the output being the label or answer). In this experiment, the dataset had around 10,000 data items. The model will use the dataset to adjust its parameters to lower its global loss function. The goal of supervised learning is to have a model which is capable of making inferences with unknown data. This kind of learning method is mostly used with a classification problem or regression problem. This study, for example, is concerned with a classical classification problem because we are asking the machine learning model to make inferences with new datasets, aimed at ascertaining whether a dataset represents hybridized DNA or non-hybridized DNA.

3.0.2 Unsupervised Learning

"Unsupervised learning", by contrast, means that there is no label for the training dataset or, in other words, we will not provide any answers or explanations for the given dataset. With this learning method, the algorithm will try to ascertain the relationships between the datasets given and make a category based on the characteristics observed from the datasets. Unsupervised learning consists in categorizing or clustering the data based on its characteristics.

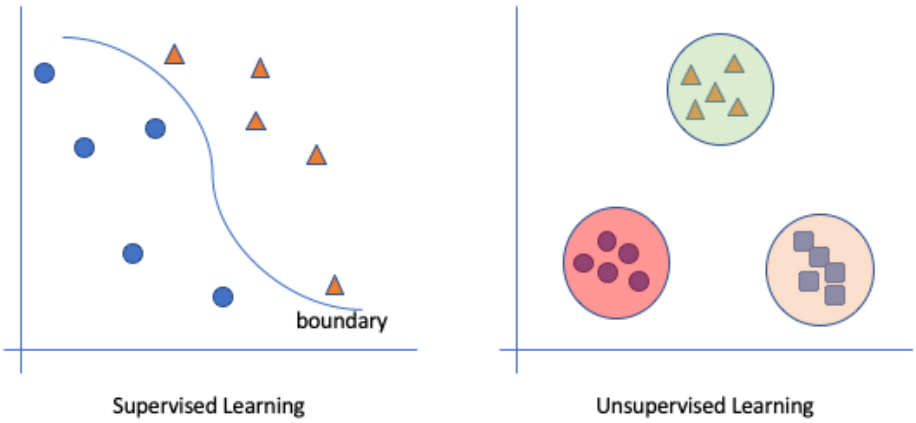

Figure 3.2: Illustrations of Supervised Learning and Unsupervised Learning

3.1 Classification

Classification is the process of selecting an object to be named or regarded as part of a certain group or category according to established criteria. This definition of classification applies in machine learning, in which features can be extracted from the given training data and these can be considered as the criteria of the classification. The labels associated with the training data are the categories or classes of the object, and the extracted criteria will be given to the machine learning algorithm as an input.

It needs to be pointed out here that the user will not tell the computer what the criteria of the classification is. Rather, the algorithm will read the data and try to formulate the classification criteria. The algorithm will keep adjusting its coefficient until it finds a minimal cost value, which will result in a model that can be used to make inferences with future datasets. The classification algorithm is intended for finding a model which can make an inference regarding which of a set of categories an unknown dataset belongs to, on the basis (or criteria) of a known dataset (training data) containing the features whose category

membership is known.

There are many classification algorithms that exist in Machine Learning. Some examples are the naive bayes classifier, decision tree classifier, Gradient Boosting Classifier, k-nearest neighbor classifier, and logistic regression. However, due to the time constraints of the research presented here, we were not be able to study all of them. Consequently, this study only concerns itself with the "n-folds cross validation" method to select two algorithms with the most accurate mean scores for the further study. The n-folds cross validation method will be described in detail later. Following the results shown in figure 3.5, this thesis will also select the k-nearest neighbor classifier as well as logistic regression for further study. The selection was made using the statistical analysis of the results we have, and does not imply that the rest of the algorithms are necessarily unsuitable.

3.1.1 K-Nearest Neighbor Classifier

The K-Nearest Neighbor, also known as the KNN, is a classifier that classifies data based on the classification of its neighbors, where K means the number of nearest neighbors to include in the voting process. The hyperparemeter for the KNN will be the number of neighbors, or the value of K. The aim is to discover what value of K will give the highest accuracy with the unknown dataset. [7]

One characteristic of the KNN model is that it is comparatively easy to build. There is no computation being done during the creation of the KNN Model in virtue of the fact that the only thing the model does as it is being created is record the data. The computation is done at the time of inference. The Euclidean distances are computed according to both the new and known datasets and the result will be recorded. Then the distances will be compared to each other and the model will use the smallest N value as its reference, where N is the number of neighbors specified. Lastly, the inference of the KNN will be based on the classification of its N-nearest neighbor.

The KNN is a straightforward algorithm, but it is also an algorithm which will suffer from the "curse of dimensionality". The main reason for this is because in the higher dimensions,

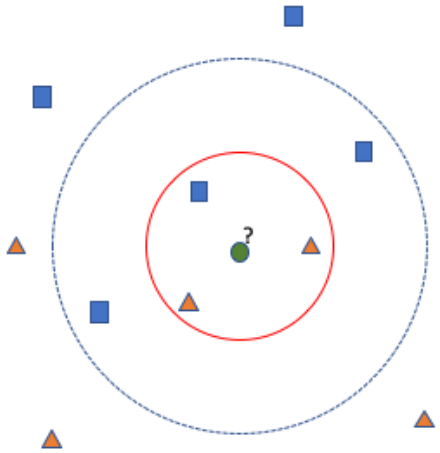

Figure 3.3: An example of KNN classification. The unknown data (classified as and represented by the green circle) should be classified as and represented by an orange triangle if K = 3 or the blue squares if K = 5.

the distance metric used in the KNN becomes meaningless. On top of that, the computation time for the distances is also longer due to the high dimension.

When making classifications with the KNN classifier, a new instance is predicted by a majority vote according to the category of the training instances of its k- nearest neighbors. Since the feature space of the k-nearest neighbor model is generally an n-dimensional real number vector, the distance is usually calculated using the Euclidean distance.

The key is the selection of the k value. If the k value is too small, it means that the overall model becomes complicated and over-fitting can easily occur. Over-fitting refers to the model fitting too closely to the data used for its training and consequently losing its flexibility for inputting unknown data in the future. That is, if the adjacent instance point happens to be noise, the prediction will be wrong.

In extreme cases, according to k=1 (called the nearest neighbor algorithm), for a point x to be predicted, the point closest to x determines the category of x. The increase in k means that the overall model becomes relatively simple. In extreme cases, such as k = N, the model

will be too simple for it to predict that k belongs to the class with the most training sets, regardless of the input instance. The experience is that the value of k generally becomes relatively small value, and a cross-validation method is usually adopted to select the optimal k value.

When implementing the k-nearest neighbor classifier, the main consideration is how to perform a quick search for the k-nearest neighbor using the training data, which is especially important when the dimension of the feature space is large and the training data capacity is large.

The easiest method for implementing the k-nearest neighbor method is linear scanning, in which the distance between the unknown data and each data point in the training dataset is calculated. When the training dataset is large, the calculation is very time consuming, and the method is rendered unfeasible. In order to improve the efficiency of the search, it is worthwhile to use a special structure, such as the k-dimensional tree, to store training data and reduce the number of calculation distances.

3.1.2 Logistic Regression

Logistic regression is a classification model which has an output of dichotomous variables, which means that its output can only have two possible values, such as 0 and 1, or "Yes" and "No". Logistic regression is best for analyzing a dataset when there are one or more independent variables that determine an outcome, which may make it suitable for the dataset of this study. One of the hyperparameters we will take a look at is the penalty term, or regularization. When a regression model uses L1 as its penalty or regularization technique, the model is known as "lasso regression". On the other hand, the model is known as "ridge regression" when it uses L2 as its regularization technique. [7] [8]

The difference between the two kinds of regression is their cost function. In lasso regression, the cost function is defined as follows:

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3.3}$$

"Lasso regression," also known as the "least absolute shrinkage and selection operator", adds an absolute value of magnitude of the coefficient as a penalty term to its lost function. This operation enhances the model's accuracy of inferences and its interpretability.

"Ridge regression" is defined as follows:

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{3.4}$$

The squared magnitude of the coefficient is added as its penalty term to the loss function.

There is λ in both lasso regression and ridge regression. If is zero, we get "ordinary least squares," which is a type of linear least squares method for estimating the unknown parameters in a linear regression model. On the other hand, if we have a high value for , then it will add too much weight and this will lead to under-fitting. The main difference between the two regressions is that lasso regression shrinks the coefficients of the least important features to zero and, consequently, removes some features altogether. This method works well for a dataset which has a huge number of features, such as the dataset used in this thesis. Finally, it needs to be said that while ridge regression includes all the features to prevent overfitting, it is not very useful in cases of a high number of features.

3.2 Clustering

Clustering, as an unsupervised learning algorithm, aims to group objects based on how close they are to each other. This also means that the objects in the same group (or cluster) are more similar in some way to each other than to those in other groups or clusters. Even so, clustering will not be investigated further here because the algorithm cannot help us with our goal. One of the major reasons is because we cannot control how many clusters will be formed by the algorithms. The problem we want to solve in this study only requires the two

values of hybridized (1) or non-hybridized (0), whereas clustering may produce more than two groups.

3.3 Model Evaluation

A machine learning model can be evaluated in many ways, including "classification accuracy," "logarithmic loss," the "confusion matrix," "area under curve," "F1 score," the "mean absolute error," and the "mean squared error". In this section, we will briefly outline and evaluate these methods in the pursuit of finding out which is suitable for us.

Classification accuracy looks at how accurate the inference made by the model is, referring to the ratio of the number of correct predictions to the total number of input samples.

$$\text{Accuracy} = \frac{\text{Number of correct prediction}}{\text{Total number of predictions made}} \tag{3.5}$$

The accuracy will work well only if there is an equal number of samples for each class. In other words, for a binary classification problem such as the one treated in this study, we need 50% of the data classified as class A and another 50

Logarithmic loss works by penalizing false classifications. This is suitable for multi-class classifications and it is defined as follows:

$$\text{Logarithmic Loss} = \frac{-1}{\text{N}} \sum_{\text{i=1}}^{\text{N}} \sum_{\text{j=1}}^{\text{M}} \text{y}_{\text{ij}} * \log(\text{p}_{\text{ij}}) \tag{3.6}$$

where

y_{ij}, indicates whether sample i belongs to class j and

p_{ij},, indicates the probability of sample i belonging to class j.

In general, a smaller logarithmic loss indicates a higher accuracy for the classifier.

A **confusion matrix** is a table used to describe the performance of the classifier for which the true values are known.

To reiterate, the problem which is being treated in this study is a binary classification problem. We have samples which either belong to Hybridized or Non-Hybridized, and we have our classifier inferring the class of each sample. This gives the following results:

data size = 165	Predicted Hybridized DNA	Predicted Non-Hybridized DNA
Actual Hybridized DNA	50	10
Actual Non-Hybridized DNA	5	100

Table 3.2: Sample Confusion Matrix

The accuracy can then be computed as follows:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{False Negative}}{\text{Total Sample}} \tag{3.7}$$

Based on the formula given above, the accuracy of the sample data we have in Table 3.2 is 0.91. Naturally, a higher value indicates a better accuracy.

The **area under curve**, also known as the AUC, is a commonly used method for evaluating a machine learning model. But before talking about the AUC, we first need to understand the following concepts:

- True Positive Rate, also known as "Sensitivity," is defined as follows:

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negative}} \tag{3.8}$$

- False Positive Rate, also known as "Specificity," is the ratio of negative data items that have been incorrectly predicted to all negative data points. It is defined as follows:

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negative}} \tag{3.9}$$

The values of the true positive rate and false positive rate will fall within the range of 0,1, and both values will be computed at threshold values, such as 0.00, 0.02, etc., up to 1.00. Following this, a graph will be drawn based on the result of the computation. The

plot of the positive rate against the false positive rate at different points in 0, 1 and the area under curve is the area under the curve plotted using these values.

The F1 score is a measure of a test's accuracy. It considers both the precision and the recall of a test in computing the score.

Precision, as defined before, is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positive}} \tag{3.10}$$

Recall is the number of correct positive results divided by the number of all samples that should have been identified as positive:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negative}} \tag{3.11}$$

Finally, the F1 Score will be computed as follows:

$$\text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.12}$$

The mean absolute error is the average of the difference between the predicted value and expected value. This value gives us a measure for how far the predictions were from the expected output. However, this value does not give any further information with regard to our model. The mathematical formula for the mean absolute error is:

$$\text{Mean Absolute Error} = \frac{1}{\text{N}} \sum_{\text{i}=1}^{\text{N}} |\text{y}_\text{i} - \hat{\text{y}}_\text{i}| \tag{3.13}$$

The mean squared error is similar to the mean absolute error. The only difference between the two is that the latter computes for the sum of the absolute value of the difference between predicted and expected values, and the former computes for the sum of the squared value of

it.

$$\text{Mean Squared Error} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.14}$$

In this study, we will be using the accuracy of the classification as our matrix for evaluating the model since we are certain that 50% of our dataset is composed of Hybridized DNA data and 50% is composed of Non-Hybridized DNA data.

3.4 Cross Validation Mean Score

The problem addressed in this study is a typical classification problem which can be solved using machine learning. There are many classification algorithms available in machine learning, such as the cross validation mean score, the Naive Bayes classifier, the Gradient Boosting classifier, and logistic regression. The question now is to ascertain which is the most suitable for solving the problem treated in this study.

The cross validation mean score is another common method used to validate machine learning algorithms. In this study, the cross validation mean score will be used to perform an initial evaluation of the selected algorithms and as a method to select the best hyperparameter of the algorithm. However, the first thing to do is to evaluate the algorithms in question.

The fastest way to determine how accurate an algorithm is by considering its cross validation mean score. The algorithm behind it is quite simple; the given data will be divided into N parts, and each time, the cross validation method will take the N - 1 part as training data to train a machine learning model and use the remaining part to validate the model which is built. The accuracy score will be computed for each validation cycle, and the Nth validation is completed, an average score will be computed. This algorithm will return a mean score of the accuracy which can be used for evaluating the selected models.

In this study, a 10-fold cross validation will be implemented together with the following classification algorithms:

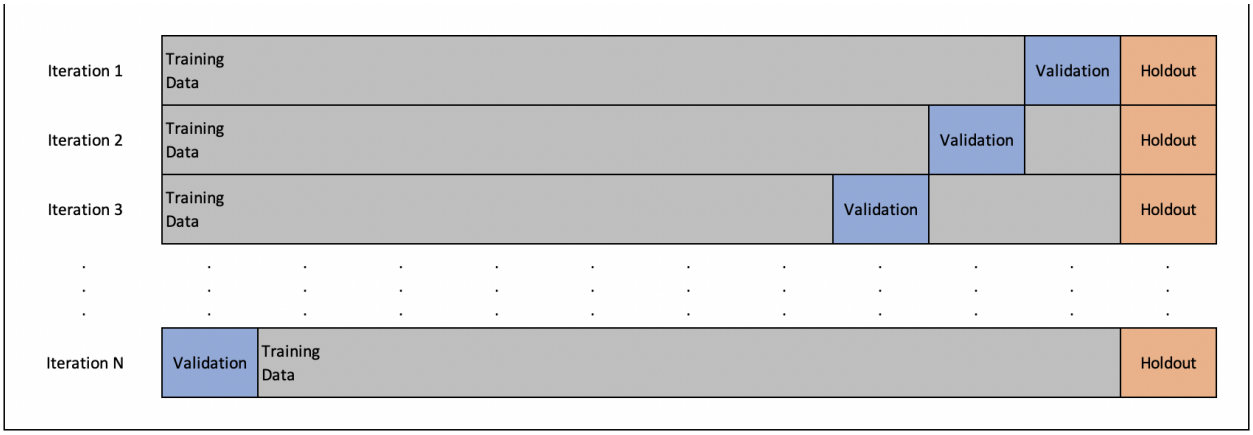

Figure 3.4: Diagram of N-fold Cross Validation

- Bernoulli Naive Bayes Classifier
- Decision Tree Classifier
- Gaussian Naive Bayes Classifier
- Gradient Boosting Classifier
- K-Nearest Neighbor Classifier
- Logistic Regression
- Random Forest Classifier
- SVM Linear Regression Classifier

The data used in the 10-fold cross validation will not undergo any data processing, meaning that the original data will be used in the validation. Additionally, the algorithms are not fine-tuned, meaning that all the algorithms will be used with their default hyperparameter. The main reason for this is to maintain the fairness of the validation. Consequently, the result we have may have been different if we performed fine tuning.

Figure 3.5 shows the result of the 10-fold cross validation using the original dataset. It is clear that the k-nearest neighbor classifier had the highest score among all the classifiers. Even so, the highest accuracy score, which is 64.32%, is still low if the biosensor's domain

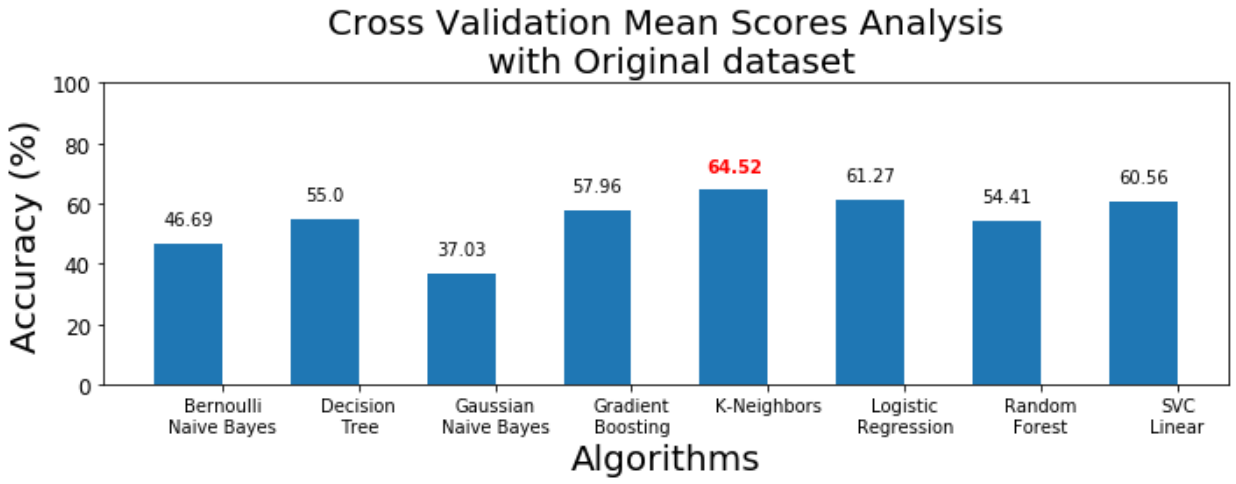

Figure 3.5: The results of the 10-Fold Cross Validation

of application (e.g. biomedicine or forensic science) is taken into consideration, and as mentioned, this score may not be the best result for the purposes of solving the problem with which we are concerned. Nevertheless, it can serve as a baseline for the future solution. The question now will be: which hyper-parameter can improve this score?

Based on the results shown in Figure 3.5, the top two algorithms—the K-nearest neighbor classifier and logistic regression—will be considered for further investigation.

It must be remembered that the data we used for the cross validation is the original data we received from the biosensor. In other words, this data did not undergo any processing or feature extraction. However, the feature engineering method will be implemented before we investigate further.

3.5 Neural Networks

The classic machine learning algorithms discussed in the previous sections utilize the statistics to learn the features from the known data and attempt to infer the appropriate classification of the unknown data. Deep Learning, however, utilizes neural networks, also known as "artificial neural networks," for the computation, extraction and abstraction of the data's features.

Unlike classical methods, neural networks are designed for the purpose of looking for the patterns in the data. This network can interpret, label or cluster the given data. The neural network is built up from artificial neurons, like in Figure 3.6, which imitate biological neural networks such as human brains, to learn the patterns or features from the known dataset and make an inference with an unknown dataset. In this section, we will briefly review some of the artificial neural networks, namely shallow neural networks and deep neural networks (DNN), to see if neural networks can help us in this matter and how far can we go with them.

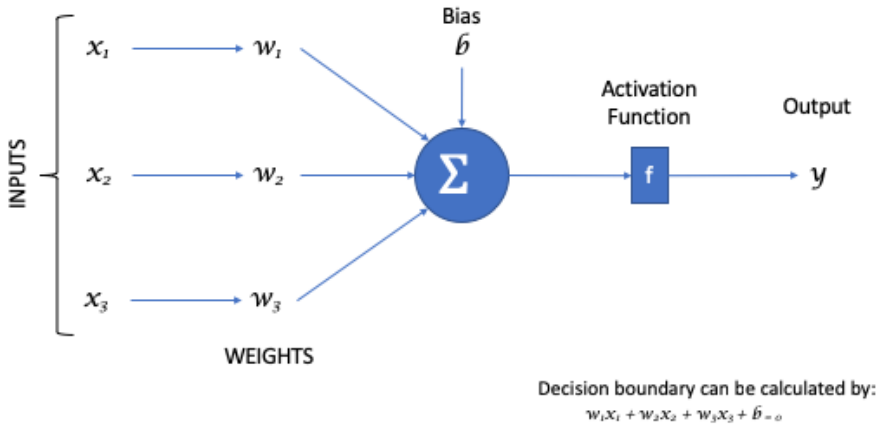

Figure 3.6: An artificial neuron

At the most basic level, neural networks are composed of 3 layers: the input layer, hidden layer, and an output layer (see Figure 3.8). An input layer is an entrance of the features. It only serves as a receiver and no computation is performed using it. The number of input neurons is equivalent to the number of features you have with your dataset.

The hidden layer is between the input layer and the output layer, and does the "thinking" or "computing" by means of a multitude of neurons. Each neuron receives inputs and fires an output. As illustrated in Figure 3.6, a neuron will receive an input x, together with the

weight w of each. The product of the inputs and their respective weights will be summed up together with a bias b. The activation function will take the weighted sum and compare it to a threshold of the neuron. If the weighted sum of the input exceeds the threshold of the neuron, then the activation function will fire a signal. If the sum does not exceed the neuron's threshold, the activation will not fire a signal.

A hidden layer is composed of the number of neurons and layers the user specifies. Each layer will be connected to the next hidden layer, or the output layer if that layer is the last layer of the hidden layer. An output layer is simply an output of the whole network. It receives an input from the hidden layer and performs the last computation to decide whether or not to fire a signal as the final result of the network.

3.5.1 Shallow versus Deep Neural Networks

A shallow neural network is a neural network with only one hidden layer, as opposed to a deep neural network which will have several hidden layers. There is no evidence as to whether shallow neural networks are better than deep neural networks or that deep neural networks are better than shallow neural networks. Consequently, this study will be concerned with an experiment to see which type of network can generate the best result. Of course, the bigger the network, the more computation and computational power are needed.

A deep neural network, as opposed to a shallow neural network, is a network with multiple hidden layers. Those layers are used for feature extraction and transformation. Each successive layer uses a previous layer's output as its input. In this structure, each layer attempts to learn to transform its input data into a slightly more abstract and composite representation before passing it to the next layer.

In this study, the deep neural network will be built based on the result found in the shallow neural network. The best results will be selected from the experiment using the shallow neural network and the succeeding layer will be added accordingly. The ultimate goal of this experiment is to discover how many hidden layers and neurons it takes to achieve the best accuracy score. Due to limitations of computing power, however, we need to avoid

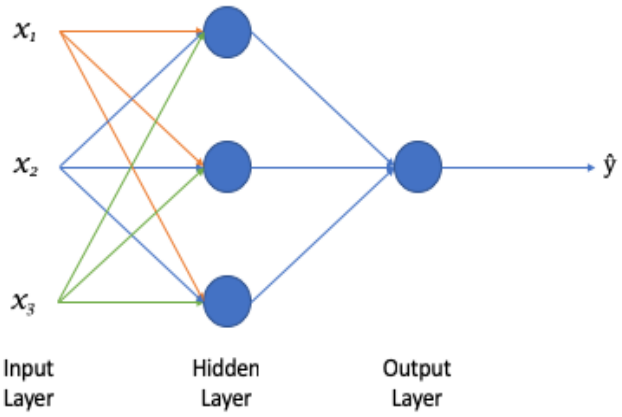

Figure 3.7: An illustration of a shallow neural network

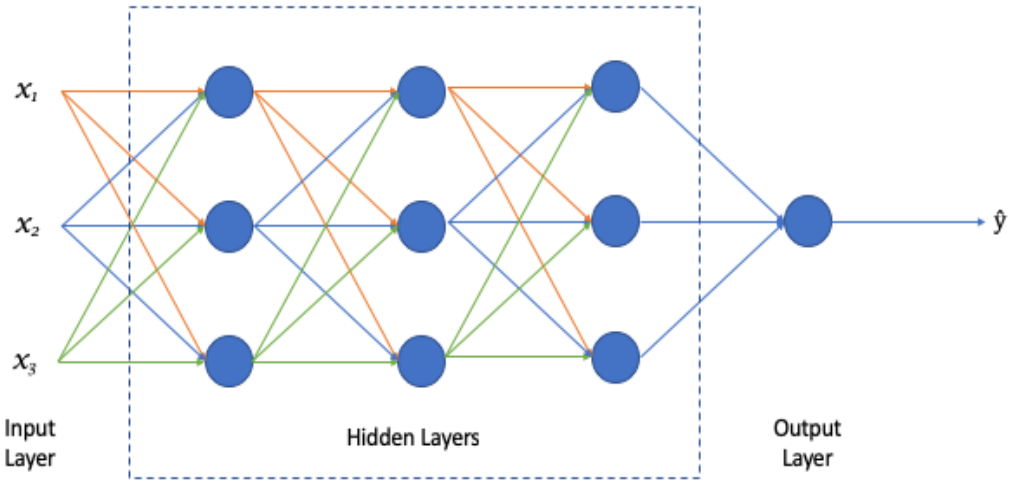

Figure 3.8: An illustration of a deep neural network

building a big network and, with this in mind, the smallest first layer neuron, which is 100, will be selected as our first layer and we will expand it accordingly.

Chapter 4

The Dataset and the Application of Feature Engineering

4.1 The Dataset

The dataset used in this study was recorded using the CNT-based biosensor developed by Dr. Sinha's Team [2]. What follows is a description of the procedure followed for the experiment.

Before commencing each experiment, two mixtures need to be prepared. The first is the pure TE Buffer solution, which will act as the control for the experiment. The other mixture contains the single-stranded DNA.

A device called an "8-Channel Pipette", figure 2.2, was used by the research team to make sure that only one drop of each solution will be injected into its corresponding chip at the same time. The biosensor experiment, i.e., the measurement of the electrical current passing through the DNA solution, will be triggered only after the mixtures are prepared.

The sensor will record the voltage right after the mixtures are put onto the chip. Additional samples of the mixtures will be added five minutes after the first drop, and the sensor will continue to record for another five minutes. The sensor will therefore be measuring and recording the voltages for ten minutes in total. There will be 1,000 records for every minute. There are a total of 10,000 voltages recorded in ten (10) minutes, and so 10,000 voltages will

be collected for each experiment.

The data of the four chips will be recorded separately, and each chip will have its own file for the record. Each experiment will generate a folder containing four files, each with two columns recording the time and the voltage read from its corresponding chip as follows:

- CHIP 0: This is data recorded from chip 0, which contains the TE Buffer solution only. This will be used as a reference for the other chips.
- CHIP 1 - CHIP 3: This is all the data recorded from chips 1, 2, and 3 respectively. The solution given to these chips is the mixture which contains the single-stranded DNA. These chips will capture the changes to the voltage in the solutions and report them back as a text file.

Past research[2] utilized chips which measured DNA and carbon nanotubes and computed an average voltage across three chips for each recorded voltage as their final dataset. The result of this computation was used as the final dataset in determining whether hybridization took place or not. However, due to the scarcity of available data, we decided to treat each chip as an individual sample, meaning that there is no averaging of the three chips' dataset. In this way, more data can be utilized in the process of training and testing as part of machine learning, which is one focus of this study. In total, 138 samples are available for use in our data processing algorithm.

The manner in which past research teams[2] have handled the data also needs to be noted. Before the start of the experiment, the average voltage was computed prior to the normalization process (see the definition in equation 4.1). In this way, all the datasets will start with the same voltage value of 1. However, the normalization method implemented here was not the same as the usual statistical normalization methods such as "feature scaling," the "coefficient of variation," and a "standardized moment." The normalization used by Dr. Sinha's team is defined as follow:

$$x_i^{\text{normalized}} = x_i^{\text{un-normalized}} - x_i^{\text{chip0}} \tag{4.1}$$

for i = 1, 2, ..., N , where N is the size of the feature dataset we have. In the original measurement [1] x0 is chosen to be the first record of the voltage; i.e., x0 = 1 after normalization. As will be shown later, this normalization method did contribute to an improvement in machine learning performance.

4.2 Feature Engineering

Researchers of machine learning spend more than half of their time finding data characteristics to feed to the machine learning model. This finding or extracting of data characteristics is called "feature engineering." It is very rare that a machine learning model can be successful if there is no feature engineering using the data. In this study, different approaches will be implemented to extract features from the data.

One reason why feature engineering is needed is because the computer cannot understand non-numerical data. For example, in the sample data set, Table 3.1, the computer cannot understand the difference between "male" and "female" under the "Sex" column, and the same thing will occur in the column "Embarked." Given this situation, the computer cannot comprehend the string data or do any of the necessary computation. Consequently, we need to convert the string data to something meaningful to the computer program. This could be, for example, a numeric value, with which the computer program can give it some meaning and perform its calculations. Another example is cleaning up the data, which includes filling up the missing values by means of performing complicated computations or by merely eliminating those values. The processes of converting and cleaning data are both examples of feature engineering.

One of the biggest problems when it comes to the data in this study is missing values. The data scientist spends much of their time either removing the data or replacing those missing values by means of new computations. Fortunately, this need not be the case in this experiment because the data was accurately recorded by the biosensor. There are no missing values or non-numerical figures. It must also be noted on the basis of the cross validation

mean score, however, that the original data did not contribute much to the machine learning models. We therefore we need to find another method which can help us extract the good features.

Before finding a new feature engineering method, we also tried some of the existing methods with the current dataset. One of them is known as "principal component analysis," or PCA. PCA involves statistical procedures which use an orthogonal transformation of the data to convert a dataset of possibly correlated variables into a new dataset of values of linearly uncorrelated variables known as principal components. The initial result of the PCA was relatively poor. It showed that a single dataset with 10,000 features can be reduced to only 2 variables. The reduced dataset performed poorly with the classification algorithms. The reason for this may, however, be something to do with our implementation of the PCA. Unfortunately, due to time constraints, we could not continue with the PCA to determine whether this was the case.

During the research for this study, we found the following feature engineering methods to improve the machine learning model's performance. These methods can be implemented separately or combined.

- Derivative Coefficients

  In mathematics, slopes are commonly used to describe a line considered in abstraction from its coordinates or other characteristics. Such characteristics as the coordinates, the length of the line, and the slope of the line seem irrelevant for machine learning. Consequently, a new approach to extracting a slope of two adjacent points and forming a new dataset was developed for the purposes of this study. These changes of the line are called "derivative coefficients" and are defined as follows:

$$\mathrm{m_i} = \frac{\Delta \mathrm{y}}{\Delta \mathrm{x}} = \frac{\mathrm{y_{i+1}} - \mathrm{y_i}}{\mathrm{x_{i+1}} - \mathrm{x_i}} \tag{4.2}$$

  In this formula, the first point of the dataset, i0, is set to 0. That means that the first point will be discarded in the differential vector (slope) calculations. This can be

further simplified since changes to x are always equal to 0.1. Thus, the final equation of the derivative coefficient will be defined as

$$m_i = (y_{i+1} - y_i) \tag{4.3}$$

and

$$m_0 = 0 \tag{4.4}$$

- Quantized Derivative Coefficients

  The notion of "quantified derivatives" was derived from the derivatives coefficient defined above. The slope (or derivative) can be categorized into the positive slope, zero (no slope) or negative slope. Based on this distinction, the computed derivatives were quantified as follows:

$$x_i = \begin{cases} 1 & \text{if } m_i > 0 \\ 0 & \text{if } m_i == 0 \\ -1 & \text{if } m_i < 0 \end{cases} \tag{4.5}$$

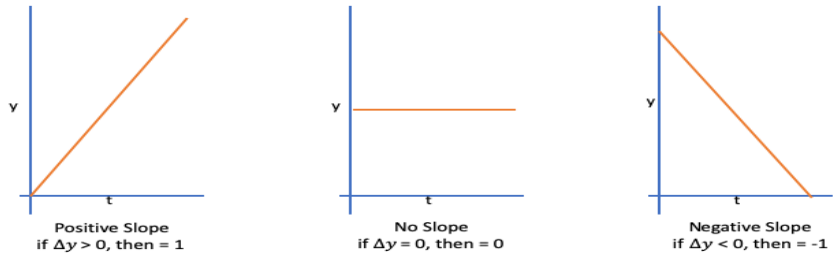

Figure 4.1: Quantized Derivative Coefficient

The results of our experiment using these data will be discussed in the next chapter. We will see how much improvement in terms of accuracy score can be achieved using these feature engineering methods.

Chapter 5

Results and Analysis

In this chapter, we will be looking at the experiments performed for this study with the following algorithms/methods:

- K-nearest neighbor classifier
- Logistic regression
- Shallow neural network
- Deep neural network

And each of them will be performed by means of the following steps:

- First, taking a look at the performance of the model using its default hyperparameter(s) with the three datasets we have.
- Using the 10-folds cross validation method to find the best hyperparameter(s).
- Creating a model using the best dataset and hyperparameter(s) and evaluating the created model.
- Create an Ensemble model using the selected algorithms with the best dataset and hyperparameter(s) and evaluate the created model

Before proceeding with the details of the results, the following points must be noted:

- The current dataset will be divided into a training dataset and a test dataset with the respective ratios of 66% and 34%. This means that the test dataset used in evaluating the machine learning model has never seen by the target model before.
- The training dataset will then be further divided into a sub-training dataset and a sub-test dataset using the N-fold cross validation method. This also means that the test set of the N-fold cross validation method is also something new to the model.

5.1 K-Nearest Neighbor Classifier

The following three datasets will be used together with the KNN to see which combination of datasets and KNN algorithms can provide better accuracy.

- Raw dataset (RD)
- Derivatives coefficient (DC)
- Quantized derivatives coefficient (QDC)

As mentioned, the cross validation method will be used to find the value of K. In this study, the value of K will range from 1 to 39, and we will only be considering the odd numbers. The reason for this is the avoidance of ties.

The result of the cross validation is shown in Table 5.1. We can see that the non-normalized dataset worked better than the normalized dataset, which means that the machine learning model can learn more features from the non-normalized dataset. It also proves one of the hypotheses to be correct. Furthermore, the feature engineering methods also appear to have worked well with the models. Finally, as the results indicate, the derivative coefficient gave more features for machine learning than the original dataset, and the accuracy score increased from 60.87% to 69.57%. The quantized derivative coefficient, appears to have made no difference when it comes to the derivatives coefficient.

Dataset	w/o Normalization	w/ Normalization
Raw Dataset (RD)	60.87%	56.52%
Derivative Coefficient (DC)	69.57%	67.39%
Quantized Derivative Coefficient (QDC)	69.57%	69.56%

Table 5.1: KNN with the default hyperparameter (K=5)

As mentioned at the beginning of this section, tuning the hyperparameter may help with increasing accuracy. In KNN, the only hyperparameter that can be adjusted is the size of K, and Table 5.2 shows the results of the tuning.

Size of K	w/o Normalization	w/ Normalization
1	61.61%	51.11%
3	**63.97**%	55.22%
5	61.97%	62.00%
7	62.08%	59.50%
9	58.64%	62.75%
11	52.14%	60.75%
13	54.39%	58.50%
15	55.64%	57.50%
17	58.64%	56.39%
19	58.64%	58.75%
21	56.64%	57.64%
23	56.64%	57.39%
25	59.86%	58.75%
27	59.86%	58.75%
29	59.86%	59.86%
31	59.86%	59.86%
33	59.86%	59.86%
35	59.86%	59.86%
37	59.86%	59.86%
39	59.86%	59.86%

Table 5.2: Cross Validation with different sizes of K

Table 5.2 shows that the non-normalized dataset had the best result (63.97%) with the neighbor size of 3 and the normalized dataset had the highest accuracy score of 62.75% using the neighbor size of 9. Using these values, an individual machine learning model and ensemble model can be built accordingly.

Table 5.3 shows the results of an individual KNN model built using a hyperparameter

with different datasets. Similar to the previous experiment using KNN without any tuning, a model using non-normalized quantized derivative coefficient datasets returned the best accuracy score, which was 86.96%. However, the result of the derivative coefficient was 84.78%, a significant increase from the previous result of 69.57%. On the other hand, the results of the experiment using the normalized dataset were consistent with those of the previous experiment. After normalization, all the datasets performed poorly compared to the non-normalized dataset.

Dataset	w/o Normalization(K=3)	w/ Normalization(K=9)
Raw Dataset (RD)	60.87%	65.22%
Derivative Coefficient (DC)	84.78%	73.91%
Quantized Derivative Coefficient (QDC)	86.96%	69.57%

Table 5.3: Individual KNN Model with Hyperparameter Tuning

One single machine learning model might not be enough to make a good inference with an unknown dataset, so it can be reasonably asked whether multiple models would suffice to resolve the issue. The idea of an ensemble of models refers to combining several machine learning models to help improve the accuracy of the learning done by the machine. In this experiment, the Bagging algorithm, also known as Bootstrap Aggression, will be implemented to see if the performance can be improved using the idea of an ensemble of models.

Dataset	w/o Normalization(K=3)	w/ Normalization(K=9)
Raw Dataset(RD)	67.39%	69.57%
Derivative Coefficient (DC)	69.57%	69.57%
Quantized Derivative Coefficient (QDC)	69.57%	69.57%

Table 5.4: Ensemble KNN Model

Table 5.4 shows the results of the ensemble model. We can see that the results did not meet our expectations. The ensemble models worked better than the initial model but still perform poorly compared to the individual model with hyperparameter tuning. We can therefore conclude that the ensemble method of the KNN is unsuitable for our needs in this study. In addition, the highest accuracy of the KNN we can have using non-normalized the quantized derivative coefficient is 86.96%. This, however, is not accurate enough for real-

life application. In the next section, we will look at another classification model—logistic regression—and see how the experiment using this model turns out.

5.2 Logistic Regression

There are two important hyperparameters in logistic regression. One is the value p, which stands for "penalty" with two possible values. One is the value "l1," which stands for lasso regression. Another, "l2," stands for ridge regression. The other hyperparameter is the value of c, which means the inverse of the regularization strength with a default value of 1.0. This study will implement both lasso and ridge regression, and so the only hyperparameter to be adjusted is the value of C. Furthermore, we are not interested with the optimizer at this state, and so the experiments performed in this study will be using the default, optimized Limited-Memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS).

As with the KNN, the logistic regression will first be analyzed using the original data (OD), derivative coefficient (DC) and quantized derivative coefficient (QDC). There will be no tuning of the hyperparameters for the training dataset and for building the model, and it will be evaluated using test dataset.

Dataset	Penalty L1, c=1.0		Penalty L2, c=1.0	
	w/o Normalization	w/ Normalization	w/o Normalization	w/ Normalization
OD	67.39%	60.86%	78.26%	58.70%
DC	69.56%	69.56%	76.08%	69.57%
QDC	69.56%	65.21%	95.65%	69.57%

Table 5.5: Accuracy score of logistic regression with default hyperparameters

Table 5.5 shows the results of the logistic regression without any tuning of its hyperparameters. As with the KNN, the normalized data performed poorly. Moreover, we notice that the model with L2 as its penalty works better than the L1 penalty model. Additionally, the logistic regression seems to have worked well with the quantized derivative coefficient, returning a 95.65% accuracy score with the L2 penalty. It is now worth asking how far this model can go.

Aside from penalty term, another hyperparameter which can be adjusted in the logistic regression is the value of c, which means inverse of regularization strength. This value of c is similar to what we find with support vector machines, wherein the smaller values specify stronger regularization. We will be trying different values for c in the next experiment in both L1 and L2. We will be attempting to find best value of c using cross validation methods.

Value of C	Penalty L1		Penalty L2	
	w/o Normalization	w/ Normalization	w/o Normalization	w/ Normalization
0.001	40.14%	40.14%	59.81%	56.22%
0.010	40.14%	40.14%	66.19%	55.75%
0.100	40.14%	57.72%	70.30%	61.36%
1.000	63.11%	55.50%	70.56%	66.22%
10.00	68.19%	56.58%	71.56%	67.33%
100.0	72.30%	58.83%	72.69%	65.08%
1000	74.67%	63.72%	70.69%	67.19%

Table 5.6: Cross Validation with different values of C

Table 5.6 shows the results of the cross validation of different values of c with the combination of non-normalized and normalized datasets. In ridge regression, the non-normalized dataset has a high accuracy of 72.69%, with a c value of 100. On the other hand, the normalized dataset performed well with a c value of 10 and accuracy score of 67.33%. In lasso regression, we have 1000 as the value of c for both the non-normalized and normalized datasets. Based on this result, further investigations will utilize the combination shown in Table 5.7 below.

	Penalty L1		Penalty L2	
	w/o Normalization	w/ Normalization	w/o Normalization	w/ Normalization
Value of C	1000	1000	100	10

Table 5.7: Hyperparameter Combination

Table 5.8 shows the results of logistic regression models (with hyperparameter tuning) using the test dataset. As we observed, the model using the Normalized dataset exhibits no differences after hyperparameter tuning. On the other hand, we can see a significant improvement when it comes to the Non-Normalized Dataset. However, the best accuracy score

is 95.65%, which is the same as that of the previous experiment using the non-normalized quantized derivative coefficient.

Dataset	Penalty L1, C=1000		Penalty L2, C=100/10	
	w/o Normalization	w/ Normalization	w/o Normalization	w/ Normalization
OD	89.13%	60.87%	84.78%	58.70%
DC	86.95%	69.57%	84.78%	71.74%
QDC	84.78%	65.22%	95.65%	69.57%

Table 5.8: Logistic Regression with the best value of C

The ensemble method, using the Bagging Classifier of the scikit-learn library, was also implemented with the Logistic Regression. We specified the following parameters to the Bagging Classifier:

- Base classifier = Logistic Regression with the best value of C found
- Number of estimators = 100
- Maximum sample size = 30

"Max Sample" refer to the number of data items that will be used for training a classifier. In this study, a Max Sample of 30% was used to make sure that the models generated will be different from one another.

As we can see in Table 5.9, the ensemble method did not work well with the original dataset and derivative coefficient. It did, however, work well with the quantized derivative coefficient dataset. The ensemble method with lasso regression results in an accuracy of 97.83%. Ridge regression maintains the same level of accuracy at 95.65

Dataset	Penalty L1, C=1000		Penalty L2, C=100/10	
	w/o Normalization	w/ Normalization	w/o Normalization	w/ Normalization
OD	67.39%	65.22%	69.57%	58.70%
DC	76.09%	73.91%	78.26%	65.22%
QDC	97.83%	67.39%	95.65%	63.04%

Table 5.9: Ensemble of Logistic Regression

At this point, we have already found a model which can achieve 97.83% accuracy with the test dataset without using any deep learning method. Yet, taking into account the real-world application of the biosensor, 97.83% accuracy may still not be high enough. In the next section, we will look at two basic neural network methods to ascertain whether they can achieve a higher rate of accuracy than what we have right now.

5.3 The Shallow Neural Network

As in classic machine learning experiments, we will be using different datasets derived using combinations of feature engineering methods and with different sizes of neurons. First, the shallow learning will be implemented with the dataset. We will start with 100 neurons in the hidden layer and increase the number of neurons to 5,000. We want to discover which combination of neurons and datasets give us the best result.

Table 5.10 below shows us the results of a shallow neural network built using the training dataset and evaluated using test dataset. As we can see, the best accuracy we can achieve here is still 97.82% by using the non-normalized quantized derivative coefficient with 100, 600, 2,500, and 4,000 neurons. As with classic machine learning methods, the non-normalized dataset performs better than the normalized dataset. Consequently, we will only be focussing on the non-normalized quantized derivative coefficient in the succeeding experiments.

Based on the information gathered from the experiment, we can add hidden layers to perform the computation, abstraction or extraction of the data features, which allows for the formation of the deep neural network after the first layer. By adding those hidden layers, the neural network should be able to extract more features with the given dataset.

However, we cannot try every possible number, and so starting with a shallow neural network might be a good way to get an idea of which one is best to continue with. The goal is to see if we can improve the model's accuracy by adding one or more hidden layers.

# Of Neurons	w/o Normalization			w/ Normalization		
	OD	DC	QDC	OD	DC	QDC
100	73.91%	86.96%	**97.82**%	67.39%	76.09%	65.22%
200	71.74%	86.96%	93.48%	60.87%	76.09%	69.57%
300	71.74%	86.96%	93.48%	73.91%	73.91%	73.91%
400	71.74%	86.96%	95.65%	67.39%	76.09%	71.74%
500	73.91%	86.96%	95.65%	67.39%	73.91%	67.39%
600	76.09%	86.96%	**97.82**%	73.91%	76.09%	71.74%
700	78.26%	86.96%	95.65%	69.57%	76.09%	73.91%
800	73.91%	86.96%	93.48%	69.57%	73.91%	73.91%
900	71.74%	86.96%	93.48%	73.91%	76.09%	74.80%
1000	76.09%	86.96%	93.48%	76.09%	76.09%	73.91%
1500	76.09%	86.96%	91.30%	69.57%	76.09%	63.04%
2000	76.09%	86.96%	95.65%	73.91%	76.09%	65.22%
2500	76.09%	86.96%	**97.82**%	71.74%	73.91%	69.57%
3000	78.26%	86.96%	89.13%	67.39%	78.26%	71.74%
3500	78.26%	86.96%	91.30%	69.57%	73.91%	76.09%
4000	76.09%	86.96%	**97.82**%	67.39%	76.09%	71.74%
4500	76.09%	86.96%	93.48%	69.57%	76.09%	80.43%
5000	76.09%	86.96%	95.65%	73.19%	76.09%	71.74%

Table 5.10: The result of the Shallow Neural Network

5.4 The Deep Neural Network

The deep neural network will be built based on the results gained from the shallow neural network. The best results are to be selected from the shallow neural network experiment and the succeeding layer will be added accordingly. To reiterate, the purpose of this experiment is to discover the number of layers and neurons which can achieve the best accuracy score. However, due to limitations of computing power, we need to avoid building a big network, and with this in mind, the smallest number of first layer neurons, which is 100, will be selected as our first layer and we will expand it accordingly.

Table 5.11 shows the results for the neural network with 2 hidden layers. As recorded, the best accuracy achieved here is still 97.82%, which means that only one test data was predicted wrongly with the second layer consisting of 20, 30 and 100 neurons. We will be using this figure to expand the network to a third hidden layer and see what will happen.

Number of Neurons		Non-Normalized QDC Dataset
1st Layer	2nd Layer	
100	10	93.48%
100	20	**97.82**%
100	30	**97.82**%
100	40	93.48%
100	50	93.48%
100	60	91.30%
100	70	95.65%
100	80	95.65%
100	90	91.30%
100	100	**97.82**%

Table 5.11: The result of 2-Layers Deep Feedforward Neural Network

Tables 5.12, 5.13, and 5.14 showed the results of the experiments performed with different sizes of the third hidden layer. The best accuracy score that can be had is still 97.82%. In addition, we also observed that in some experiments, particularly the model with 100, 30, or 24 hidden layers had a chance of 100% accuracy. The result, though, was not consistent every time, and, in fact, varied between 97.82% and 100%. Nevertheless, this model may have a good chance of achieving an accuracy rate of 100% if it is expanded further. With this goal in mind, the model is chosen for expansion to include a fourth hidden layer.

# of Neurons per Layer			Accuracy	# of Neurons per Layer			Accuracy
1st	2nd	3rd		1st	2nd	3rd	
100	20	2	89.13%	100	20	12	93.48%
100	20	3	**97.82**%	100	20	13	95.65%
100	20	4	**97.82**%	100	20	14	95.65%
100	20	5	84.78%	100	20	15	91.30%
100	20	6	95.65%	100	20	16	95.65%
100	20	7	93.48%	100	20	16	95.65%
100	20	8	91.30%	100	20	17	91.30%
100	20	9	93.48%	100	20	18	93.48%
100	20	10	95.65%	100	20	19	95.65%
100	20	11	**97.82**%	100	20	20	89.13%

Table 5.12: The result of 3-Layers Deep Feedforward Neural Network 1

# of Neurons per Layer			Accuracy	# of Neurons per Layer			Accuracy
1st	2nd	3rd		1st	2nd	3rd	
100	30	2	89.13%	100	30	16	**97.82**%
100	30	3	95.65%	100	30	17	**97.82**%
100	30	4	95.65%	100	30	18	95.65%
100	30	5	93.48%	100	30	19	**97.82**%
100	30	6	**97.82**%	100	30	20	95.65%
100	30	7	93.48%	100	30	21	**97.82**%
100	30	8	93.48%	100	30	22	**97.82**%
100	30	9	**97.82**%	100	30	23	**97.82**%
100	30	10	**97.82**%	100	30	24	**97.82**%
100	30	11	**97.82**%	100	30	25	**97.82**%
100	30	12	95.65%	100	30	26	93.47%
100	30	13	95.65%	100	30	27	95.65%
100	30	14	95.65%	100	30	28	95.65%
100	30	15	**97.82**%	100	30	29	**97.82**%

Table 5.13: The result of 3-Layers Deep Feedforward Neural Network 2

Number of Neurons			Non-Normalized QDC Dataset
1st Layer	2nd Layer	3rd Layer	
100	100	10	93.48%
100	100	20	91.30%
100	100	30	93.48%
100	100	40	95.65%
100	100	50	**97.82**%
100	100	60	91.30%
100	100	70	89.13%
100	100	80	93.47%
100	100	90	95.65%
100	100	100	95.65%

Table 5.14: The result of 3-Layers Deep Feedforward Neural Network 3

There are valid and useful suggestions by researchers on how to compute the right number of hidden layers and neurons that need to be taken into account. One is that the hidden layer should look like the following[7]:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.

Number of Neurons				Non-Normalized QDC Dataset
1st Layer	2nd Layer	3rd Layer	4th Layer	
100	20	4	1	91.30%
100	20	4	2	86.96%
100	20	4	3	80.43%
100	20	4	4	86.96%
100	20	4	4	86.96%

Table 5.15: The result of 4-Layers Deep Feed-forward Neural Network - 1

Number of Neurons				Non-Normalized QDC Dataset
1st Layer	2nd Layer	3rd Layer	4th Layer	
100	30	24	2	86.96%
100	30	24	3	93.47%
100	30	24	4	95.65%
100	30	24	5	**100.00**%
100	30	24	6	91.30%
100	30	24	7	95.65%
100	30	24	8	97.82%
100	30	24	9	97.82%
100	30	24	10	93.48%
100	30	24	11	95.65%
100	30	24	12	97.82%
100	30	24	13	93.48%
100	30	24	14	95.65%
100	30	24	15	97.82%
100	30	24	16	95.65%
100	30	24	17	93.48%
100	30	24	18	97.82%
100	30	24	19	97.82%
100	30	24	20	93.48%
100	30	24	21	93.48%
100	30	24	22	97.82%
100	30	24	23	**100.00**%

Table 5.16: The result of 4-Layers Deep Feed-forward Neural Network - 2

- The number of hidden neurons should be two thirds the size of the input layer plus the size of the output layer.

Number of Neurons				Non-Normalized QDC Dataset
1st Layer	2nd Layer	3rd Layer	4th Layer	
100	100	50	5	89.13%
100	100	50	10	89.13%
100	100	50	15	91.30%
100	100	50	20	89.13%
100	100	50	25	95.65%
100	100	50	30	97.82%
100	100	50	35	95.65%
100	100	50	40	93.48%
100	100	50	45	97.82%
100	100	50	50	91.30%

Table 5.17: The result of 4-Layers Deep Feed-forward Neural Network

- The number of hidden neurons should be less than twice the size of the input layer.

With these rules being taken into account, the experiment with the configuration given in Table 5.18 was also performed. Its accuracy, however, only reached 91.30%. This result shows that the recommended rules do not give us a model better than or equivalent to what we have right now.

Number of Neurons				Non-Normalized QDC Dataset
1st Layer	2nd Layer	3rd Layer	4th Layer	
6668	4447	2199	1466	91.30%

Table 5.18: The result of 4-Layers Deep Feedforward Neural Network

Chapter 6

Conclusion and Suggestions for Future Research

Machine learning can be utilized in conjunction with a biosensor in many ways. With this particular biosensor, we have been interested in whether we can employ machine learning to determine whether hybridization takes place by applying different approaches to feature engineering. To recap, the main objective of this study has been to ascertain the following:

- A pattern mining technique which can help us extract a large quantity of information about pattern recognition from low-level sensor signal readings.
- A machine learning algorithm and model which can perform an inference with the given dataset pertaining to whether hybridization took place.

Furthermore, here, the neural network will also be applied as a black-box to see if it can perform better than classic classification algorithms. A shallow neuron network and a deep neuron network were built for the experiments. The first conclusion we can make concerns feature engineering. Based on the results of the experiments, we can conclude that the non-normalized dataset gave more features to machine algorithms than the normalized dataset. We cannot conclude, however, that the normalization is bad for this kind of data. It does seem to be the case that the normalized method used by Dr. Sinha's team did not

work as expected. We also found out that the derivative coefficient played a significant role in our experiments. In general, the algorithms using derivative coefficient datasets can gave us a higher accuracy rate than the one using the raw dataset. We can push the accuracy even higher by quantifying the derivatives into -1, 0, or 1, which we called the "quantified derivative Coefficient".

This study did not provide a review of all classification algorithms. Instead, the two algorithms which had the highest mean score in terms of accuracy were selected, and the 10-folds cross validation was used for this purpose. Out of 8 classification algorithms, the KNN classifier, with an accuracy mean score of 64.52%, and the logistic regression, with the accuracy mean score of 61.27%, were selected for further investigation. We did not, however, conclude that the other algorithms were unsuitable for the problem under consideration here. The top two were selected in order to limit the scope of the experiments.

The KNN was one of the algorithms with a high accuracy mean score; yet, it did not perform well in our succeeding experiments. The highest accuracy we could reach using a single KNN classifier is 86.96%, with a neighbor size of 3 using the non-normalized quantized derivatives coefficient. On the other hand, the normalized dataset could only reach 73.91% as its highest accuracy score with the derivative coefficient datasets. The ensemble method was also used together with the KNN, but again, the result was not as accurate as expected. The highest accuracy score was 69.57%.

On the other hand, the logistic regression performed better than the KNN, even as the former involves more factors to consider, such as the penalty term and the value of C. We attempted to find the best combination of hyperparameters for generating a high accuracy score. In our experiment, we found that the combination of penalty L1 and the non-normalized quantified derivatives coefficient dataset gave us 84.78% accuracy while penalty L2, with the non-normalized quantified derivatives coefficient, returned 95.65%. However, when we applied the ensemble method with the logistic regression, the best result was 97.83%. This was with the ensemble of logistic regression model using non-normalized quantified derivatives coefficient datasets. At this point, we can conclude that the classical machine

learning algorithms can give us a high inference accuracy of 97.83%. However, considering the range of practical applications of the sensor, 97.83% accuracy still seems insufficient, and we must consider the neural network.

When it came to the neural network, we tried to use a deep neural network (DNN) classifier as a black-box, and used its default hyperparameters. The hyperparameters adjusted were the number of layers and the number of neurons in each layer. First, a single layer DNN was built with different sizes of neurons ranging from 100 to 5000, which allowed us to form a shallow neural network using a similar dataset to those used in classical machine learning. The result was somehow similar with the previous experiments. A model using non-normalized quantified derivatives and neuron sizes of 100, 600, 2,500 and 4,000 returned a high inference accuracy score of 97.82%. The DNN model that followed was built based on this result. Up to four layers were built. We found that there were two models which gave us 100% accuracy with hidden layers with configurations of[100, 30, 24, 5] and [100, 30, 24, 23] respectively.

It is important to note, however, that one of the readings suggested some rules of thumb for determining the correct number of neurons to be used in the hidden layers. An experiment was performed based on these rules, but unfortunately, only returned an accuracy of 91.30%—worse than the best result we achieved with the classic machine learning model.

We must also note that the experiments and results of study only make sense with the continuous time-series dataset. This also means that there should be no missing data in the dataset. One reason for this is that the feature engineering method developed here was specifically designed to be used with the biosensor mentioned at the beginning of the study. One of the characteristics of the biosensor is its high accuracy, and so there should be no missing data.

6.1 Future Work

Despite the fact that we already found two models with an inference accuracy of 100% , there remains room for future improvement. One issue in our experiments has been the size of the features in the sample data. We have 10,000 features recorded over 10 minutes for each sample data item. This size makes our model big, inefficient. and, consequently, relatively slow.

A worthwhile question is whether we really need all these features for the machine learning algorithm to learn as intended. At this point, Dr. Sinha's team have already proved that some of the recorded data can be considered as noise and eliminated from the reading. However, the remaining number of features is still very large huge, and we believe more data can be eliminated. I would therefore like to suggest that researchers investigating this topic in the future try to ascertain which parts of the dataset are really essential for the model to make an inference.

It may be possible to reduce the size of the dataset and further improve the performance of the model. This can be done by exploring other reduction methods, such as the "principal component analysis" method, which could not be explored here due to time constraints. We also notice that the given dataset is somehow similar with the wave signals when we plot the data into a line graph. This may suggest that the noise reduction processes in wave signals constitute another possible direction for further research. By virtue of these methods, the quality of the features in the given dataset may be improved and the noise of the dataset can also be minimized.

Another thing that could be improved is the optimization methods employed. In this study, we have been using default optimization methods, such as LBFGS for the logistic regression, and the adagrad optimizer for the deep neural network. However, these might not be the best optimizers for the algorithm. Therefore, by looking at other optimizers, it may be possible to improve the performance we have achieved up to this point, with the classic classification algorithms such logistic regression

Some of the questions we set out to answer in this study have been decisively answered. We explored the quantized derivative coefficient and some of the classification algorithms, such as logistic regression, and also found that the deep neural network can work well with classification problem posed in this study. Thus, we can conclude that machine learning can be employed with the biosensor to eliminate the uncertainty of human interpretation of data. We have also found, however, that the DNN models used in this study may not be the most appropriate for the purposes which we set out. As research on biosensors is ongoing, we expect the search for the best mode to continue, using the latest data.

Bibliography

[1] R.Rettner, *DNA: Definition, Structure Discovery*, 2017. [Online]. Available: https://www.livescience.com/37247-dna.html. [Accessed: 04-Nov-2018]

[2] Sethan K. Jasti and Shawn M. McGinley and Franzel Pena and Samuel Opper and Ewa S. Kirkor and Saion K. Sinha, *Substrate Optimization of Carbon Nanomaterial Based DNA Hybridization Detection System*, Biophysical Journal, vol. 114, special issue 3, 2018. doi:10.1016/j.bpj.2017.11.3700

[3] S. Mikki, S. Sinha, C. Morris, G. Behera and Y. Antar, *A phenomenological electromagnetic theory of CNT-DNA metabiomaterials for applications to biosensing and DNA sequencing*, In: 2017 IEEE International Symposium on Antennas and Propagation USNC/URSI National Radio Science Meeting. [online] IEEE. Available at: https://ieeexplore.ieee.org/document/8072672 [Accessed 4 Nov. 2018].

[4] V.Singh, M.Kamthania, R.Pavan, S.Singh, and N.Mishra, *Biosensor Developments: Application in crime detection*, International Journal of Engineering and Technical Research, vol. stet 2014, 163-166

[5] A.Sadana, *Biosensors: Kinetics of Binding and Dissociation Using Fractals*, Elsevier Science (2003),

[6] T.Mitchell, (2013). *Machine Learning.*, New York: McGraw-Hill.

[7] Jake VanderPlas, (2016) *Python Data Science Handbook, Essential Tools for Working with Data*, O'REILLY Media, Inc.

[8] Aarshay Jain, *A Complete Tutorial on Ridge and Lasso Regression in Python*, 2016. [Online] Available: https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/ [Accessed 18 March 2019]

[9] J.Heaton, (2008) *Introduction to Neural Networks for Java*, Heaton Research, Inc.;