Machine Learning in Personalised Medicine

Waleed Al-Dabbas

Supervised by:

Prof Karen Reynolds
&
Dr. Angus Wallace

Submitted to the College of Science and Engineering in partial fulfillment of the requirements for the degree of Bachelor of Engineering (Biomedical) (Honours)/ Master of Engineering (Biomedical) at Flinders University

Abstract

Inadequate sleep is a common issue in Australia. Up to 45% of Australian adults reported a lack of sleep on a regular basis (Adams et al., 2017). This is significant given the ramifications of inadequate sleep which include obesity, type 2 diabetes and heart diseases (Adams et al., 2017). A major cause contributing to lack of sleep is sleep disorders, most commonly sleep apnea and insomnia (Adams et al., 2017). However, a large portion of clinically significant sleep disorders go undiagnosed (El-Sayed, 2012). With these disorders' high prevalence and their associated lack of screening, a single accessible and reliable tool for the identification of various sleep problems is highly needed (Senthilvel, 2011).

A sleep health questionnaire was designed by a group of health professionals to tackle this issue. However, as the questionnaire consists of over 100 questions, it is very long for users to complete which may lead to inaccuracies in responses. As such, the project aims to add machine learning tools to the questionnaire to make it more personalised and adaptable. It would also be highly desirable for the machine learning platform to allow the potential for online learning to allow further understanding of sleep disorders. This experiment is a stepping stone towards achieving that objective, with the main aim to develop a method to replicate the existing question logic using machine learning tools.

To do this, fully itemised questionnaires were used in the modeling and testing of various machine learning tools. Tests involved utilising different methods to store data into models, using various methods of representing target data attributes. And finally implementing a variety of machine learning estimators, namely neural networks, decision trees, random forest, support vector machine, k-nearest neighbour and naïve Bayes. All models were tested using 10-fold cross-validation and accuracy scores were computed accordingly.

The results showed that Gaussian naïve Bayes models achieve the highest accuracy, with neural network models being the second most accurate. Although further research is required, the outcomes of this project were promising and can be further developed and improve.

Declaration

I, Waleed Al-Dabbas, declare that the Master thesis entitled 'Machine Learning in Personalised Medicine' contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work

Recoverable Signature

X 

Waleed Al-Dabbas

Signed by: ce1f14e1-24bd-4197-ae78-2bb8aa449c2f

Acknowledgments

I hereby acknowledge the assistance provided by my academic supervisors Professor Karen Reynolds and Dr. Angus Wallace on their ongoing guidance, expertise, and support which they have provided throughout this project.

Contents

Abstract I
Declaration II
Acknowledgments III
List of Figures VII
List of Tables IX
List of Equations XI
1. Introduction 1
1.1. Background 1
1.2. The Diagnosis of Sleep Disorders 1
1.3. Proposed Project 2
1.4. What is Machine Learning? 2
1.5. Why Machine Learning? 3
1.6. Machine Learning Considerations 3
1.7. Online Machine Learning 4
2. Literature Review 4
2.1. Adaptive Questionnaires in Literature 4
2.2. Research Gap: 7
3. Methodology 8
3.1. The Existing Questionnaire 8
3.2. Existing Data 9
3.3. Data Processing 10
3.3.1. Changes Made to Questionnaire 10
3.3.2. Formatting of User Responses 10
3.3.3. Formatting of Questions 14
3.3.4. Generation of Features and Target Attributes from Raw Data 14
3.4. Predictive Modelling 18
3.4.1. Neural Network Modelling 18
3.4.2. Decision Tree Modelling 19
3.4.3. Random Forest Modelling 19
3.4.4. K-Nearest Neighbours Algorithm 20
3.4.5. Support Vector Machine Model 20
3.4.6. Naive Bayes Model 21
3.5. Testing Methods 22
3.5.1. Cross-Validation 22
3.5.2. Confusion Matrices 23

4. Results 24
4.1. Neural Network Models 24
4.1.1. Method 1 With Target Attributes in String Form 24
4.1.2. Method 2 With Target Attributes in String Form 25
4.1.3. Method 1 With Target Attributes in Vector Form 26
4.1.4. Method 2 With Target Attributes in Vector Form 27
4.2. Decision Tree Models 27
4.2.1. Method 1 With Target Attributes in String Form 27
4.2.2. Method 2 With Target Attributes in String Form 28
4.2.3. Method 1 With Target Attributes in Vector Form 29
4.2.4. Method 2 With Target Attributes in Vector Form 30
4.3. Random Forest Models 30
4.3.1. Method 1 With Target Attributes in String Form 30
4.3.2. Method 2 With Target Attributes in String Form 31
4.3.3. Method 1 With Target Attributes in Vector Form 32
4.3.4. Method 2 With Target Attributes in Vector Form 33
4.4. K-Nearest Neighbour Models 33
4.4.1. Method 1 With Target Attributes in String Form 33
4.4.2. Method 2 With Target Attributes in String Form 34
4.4.3. Method 1 With Target Attributes in Vector Form 35
4.4.4. Method 2 With Target Attributes in Vector Form 36
4.5. Support Vector Machine Models 36
4.5.1. Method 1 With Target Attributes in String Form 36
4.5.2. Method 2 With Target Attributes in String Form 37
4.6. Naïve Bayes Models 39
4.6.1. Method 1 With Target Attributes in String Form 39
4.6.2. Method 2 With Target Attributes in String Form 40
5. Discussion 41
5.1. Comparison of Models' Accuracy Scores 41
5.1.1. Method of Storing Data 41
5.1.2. Formatting of Target Attributes. 44
5.1.3. Comparison of Models 45
5.2. Comparison of Consistency in Cross-Validation Tests 46
5.2.1. Method of Storing Data 46
5.2.2. Formatting of Target Attributes. 48
5.2.3. Comparison of Models 49

5.3. Analysis of Confusion Matrices 50
6. Future Recommendations 51
6.1. Inclusion of INT Questions in Modelling 51
6.2. Implementing Scikit-learn's 'predict_proba()' Function 51
6.3. Testing Models with High Accuracy Scores Further 51
7. Conclusion 52
References: 53
Appendices 55
Appendix A: Code for Data Storage 55
Appendix B: Code for Machine Learning 58

List of Figures

Figure 1- Classification plot (left) which identifies types of cars based on their price and engine power, and a regression curve (right) which identifies the correlation between a car's mileage and its price (Alpaydin, 2010). 3
Figure 2-The format used by Miettinen et al. to represent users' data during the process of answering their adaptive questionnaire. The first column is used to identify users and the second shows questions' numbers. The remaining columns represent the possible answer choices for a question. Rows, where a single value has the value '1' and the remaining values '0', indicate that the question had been answered by the user. 5
Figure 3- A decision tree constructed by Ortigosa et al. to use for adaptive questioning. The tree was built using C4.5 algorithms and using the concept of information entropy. In the diagram, each occurrence of the letter 'Q' indicates a question. Other letters indicate possible user answers. 6
Figure 4- A hieratical structure showing the algorithm used by Kortum et al. to ask questions adaptively 7
Figure 5 – The use of nested dictionaries to store questions, answer choices to each question and vector forms of those answers. 11
Figure 6- Flow diagram showing the process of creating nested dictionaries to use in the generation of features. 12
Figure 7 - Flow diagram showing the process used to create vectors corresponding to answer choices for each question 13
Figure 8- The use of dictionaries to represent each question with a single vector. The length of each of the vectors reflects the length of the questionnaire. Since the example contains vectors which are four-elements-long, that indicates that the full questionnaire in the example is composed of 4 questions. 14
Figure 9 - Flow diagram representing the method used to construct a data frame. The contents of the data frame would later be used for the creation of machine learning models 16
Figure 10 - A diagram representing the structure of a neural network. The network is composed of input, output and a single hidden layer (" Control Systems Technology Group," n.d.). 18
Figure 11- Structure of a decision tree (DataCamp, n.d.) 19
Figure 12 - Structure of a random forest predictive model (Koehrsen, 2017) 19
Figure 13 - Representation of k-nearest neighbours 20
Figure 14 - Operation of support vector machine models (Talpur, 2017) 20
Figure 15 - Representation of 10-fold cross-validation test (Talpur, 2017) 22
Figure 16 - Representation of a confusion matrix (Narkhede, 2019) 23
Figure 17 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a neural network model trained with data stored using method 1 and target attributes in string form. 25
Figure 18 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a neural network model trained with data stored using method 2 and target attributes in string form. 26
Figure 19 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a decision tree model trained with data stored using method 1 and target attributes in string form 28
Figure 20 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a decision tree model trained with data stored using method 2 and target attributes in string form 29
Figure 21 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a random forest model trained with data stored using method 1 and target attributes in string form 31

Figure 22 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a random forest model trained with data stored using method 2 and target attributes in string form 32
Figure 23 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 1 and target attributes in string form..... 34
Figure 24 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 2 and target attributes in string form..... 35
Figure 25 -Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a support vector machine model trained with data stored using method 1 and target attributes in string form 37
Figure 26 -Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a support vector machine model trained with data stored using method 2 and target attributes in string form 38
Figure 27 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a naïve Bayes model trained with data stored using method 1 and target attributes in string form 39
Figure 28 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a naïve Bayes model trained with data stored using method 2 and target attributes in string form 40
Figure 29 - Average accuracy scores obtained from every predictive model for each of the two methods, when the target attributes were stored in string format..... 41
Figure 30 - Relative appearance of each question ID in target attributes 42
Figure 31- Average accuracy scores obtained from every predictive model for each of the two methods, when the target attributes were stored in vector format. 43
Figure 32 - Relationship between the format of target attributes and accuracy when method 1 is used to store data 44
Figure 33 - Relationship between the format of target attributes and accuracy when method 2 is used to store data 44
Figure 34-The effect of the method used to data on the consistency of predictive models when target attributes are in string format..... 46
Figure 35 - The effect of the method used to data on the consistency of predictive models when target attributes are in vector format 47
Figure 36- The effect of target attribute form on the consistency of models when method 1 is used to store the results 48
Figure 37 - The effect of target attribute form on the consistency of models when method 2 is used to store the results 48
Figure 38- Users' responses to the question enquiring about alcohol frequency 50

List of Tables

Table 1- Structure of the question manifest file which contains all the questions, their question IDs, their category and their answer choices 8
Table 2 - Correct ways to format inputs to the 'calcNextQnID' logic which contains the question logic of the existing questionnaire 9
Table 3- A table format illustration of the CSV file containing user data. The first column contains user IDs, the remaining columns each contain a question ID, and uses' responses to that question.... 9
Table 4 - The use of vectors to represent user responses. This is done so that the vectors would represent a users' features when creating a machine learning model 11
Table 5 -Desired representation of data prior to building machine learning models. Target attributes the question to be asked and they are represented in two ways as strings and as vectors. The table shows an example where the full questionnaire is composed of 4 questions. Notice that each users' responses are recorded iteratively until the questionnaire is complete. The table shows an example where only one of the questionThree or questionFour needs to be answered. 15
Table 6 – Method 2 used to store data, notice that not all questions asked to every user are stored in the dataframe. The algorithm decides which parts of the data are stored randomly. 17
Table 7 - Cross-validation results for neural network models trained with data stored using method 1 and target data in string form 24
Table 8-Cross-validation results for neural network models trained with data stored using method 2 and target data in string form 25
Table 9 - Cross-validation results for neural network models trained with data stored using method 1 and target data in vector form 26
Table 10-Cross-validation results for neural network models trained with data stored using method 2 and target data in vector form 27
Table 11 - Cross-validation results for decision tree models trained with data stored using method 1 and target data in string form 27
Table 12 - Cross-validation results for decision tree models trained with data stored using method 2 and target data in string form 28
Table 13 - Cross-validation results for decision tree models trained with data stored using method 1 and target data in vector form 29
Table 14- Cross-validation results for decision tree models trained with data stored using method 2 and target data in vector form 30
Table 15 - Cross-validation results for random forest models trained with data stored using method 1 and target data in string form 30
Table 16 - Cross-validation results for random forest models trained with data stored using method 2 and target data in string form 31
Table 17 - Cross-validation results for random forest models trained with data stored using method 1 and target data in vector form 32
Table 18 - Cross-validation results for random forest models trained with data stored using method 2 and target data in vector form 33
Table 19 - Cross-validation results for k-nearest neighbour models trained with data stored using method 1 and target data in string form 33
Table 20 - Cross-validation results for k-nearest neighbour models trained with data stored using method 2 and target data in string form 34
Table 21 - Cross-validation results for k-nearest neighbour models trained with data stored using method 1 and target data in vector form 35
Table 22 - Cross-validation results for k-nearest neighbour models trained with data stored using method 2 and target data in vector form 36

Table 23 - Cross-validation results for support vector machine models trained with data stored using method 1 and target data in string form 36
Table 24 - Cross-validation results for support vector machine models trained with data stored using method 2 and target data in string form 37
Table 25 - Cross-validation results for naive Bayes models trained with data stored using method 1 and target data in string form 39
Table 26 - Cross-validation results for naive Bayes models trained with data stored using method 2 and target data in string form 40
Table 27 - Average accuracy of models based on the method of data storage and representation of target attributes. 45
Table 28 – Range of accuracy for models based on the method of data storage and representation of target attributes. 49
Table 29 - The use of vectors to represent age ranges in datasets used for machine learning 51

List of Equations

Equation 1 - Bayes theorem 21
Equation 2 - Bayes theorem after applying the naive conditional independence 21
Equation 3 – Simplified Bayes theorem 21
Equation 4 - Classification rule application to Bayes theorem 21
Equation 5 – Gaussian/ Normal distribution 21

1. Introduction

1.1. Background

Inadequate sleep is a common issue in Australian adults (Adams et al., 2017). A 2016 Sleep Health survey showed that 33-45% of Australian adults suffer from inadequate sleep in either quality or duration (Adams et al., 2017). Inadequate sleep leads to various health risks including obesity, type 2 diabetes and heart diseases(Adams et al., 2017). It also leads to lack of productivity in daily life, a Sleep Health Foundation study estimates that lack of productivity due to inadequate sleep lead to losses of $17.9 billion in the year 2016-17 in Australia (Economics, D.A., 2017). Sleep disorders are large contributors leading to inadequate sleep in Australia (Economics, D.A., 2017). The Sleep Health Survey showed that clinically diagnosed sleep apnoea affects 8%, significant sleep insomnia 20% and restless legs 18% of adults (Adams et al., 2017). Another study shows that 5.8% of Australians suffer from excessive daily sleepiness due to sleep disorders (Motamedi et al., 2009). While these statistics include clinically diagnosed sleep disorders, many instances remain undiagnosed. It is believed that nearly 80% of men and 93% of women with clinically significant sleep apnoea have never been diagnosed (El-Sayed, 2012).

1.2. The Diagnosis of Sleep Disorders

Questionnaires are commonly used as means for the diagnosis of sleep disorders (Chervin et al., 2000; Nishiyama et al., 2014; Ohayon and Roberts, 2001; Ramachandran and Josephs, 2009). For sleep apnoea, the Berlin questionnaire and the Sleep Disorder Questionnaire are the most accurate existing questionnaires, however, they contain high rates of heterogeneity, making it possible to miss a large portion of patients (Ramachandran and Josephs, 2009). STOP and STOP-Bang questionnaires are also widely used for the diagnosis of sleep apnoea, however, due to their low specificity, they were found to lead to inaccuracies in diagnoses (El-Sayed, 2012). Sleep insomnia is commonly classified using the Diagnostic and Statistical Manual of Mental Disorders (DSM-IV) and the International Classification of Sleep Disorders (ICSD) which both were proven to lack the validity and clinical relevance to be a reliable diagnostic tool (Ohayon and Reynolds, 2009). Other self-report questionnaires have been developed to measure various aspects of sleep disturbances (Nishiyama et al., 2014). The most commonly used are the Pittsburgh Sleep Quality Index (PSQI) and the Epworth Sleepiness Scale (ESS) (Nishiyama et al., 2014). While those questionnaires were designed to measure sleep quality and subjective daytime sleepiness, they have been widely used in clinical settings with the expectation that PSQI can identify individuals with a high risk of insomnia and ESS can identify individuals with high risk for sleep apnoea (Nishiyama et al., 2014). Due to their subjectivity, they were found to be an ineffective tool to be used in a diagnostic manner (Nishiyama et al., 2014).

Past studies have suggested that primary care professionals infrequently screen for sleep disorders (Senthilvel, 2011). This is a growing concern with the high prevalence rates of these disorders, especially when considering the health concerns associated with a lack of sleep. It is apparent that a single accessible and reliable tool for the identification of various sleep problems is highly needed (Senthilvel, 2011).

1.3. Proposed Project

For the purpose of making an accessible diagnostic tool for sleep disorders, a new online questionnaire has been developed that identifies common sleep problems amongst users. The questionnaire consists of over 100 questions developed by sleep health professionals. The questions mainly focus on users' sleep habits and are mostly presented to users as checkbox format questions with 2 to 10 possible answers. It currently uses a static logical decision tree structure to dictate what questions to ask. However, as the questionnaire includes over 100 questions, completing it can be very time consuming, which may lead to the inaccuracy of users' responses in later questions. Since the questionnaire aims to identify several sleep conditions, users do not need to respond to questions that are irrelevant to their sleep conditions. As such, it is a key milestone for the project to improve its existing questionnaire by enabling it to adaptively select the questions asked based on users' responses. That way the questionnaire could be reduced dynamically to be personalised to individual users. It would also be highly beneficial to gain further understanding of sleep disorders and factors that lead to their diagnoses. With machine learning being a constantly expanding field, it is now possible to use machine learning tools to identify new patterns in datasets that are not easily identifiable by humans. Using machine learning tools to adaptively ask questions would allow achieving both objectives simultaneously. As such, this project aims to test the ability to use machine learning tools to build adaptive features to the existing questionnaire. This research was supported by the CRC for Alertness, Safety and Productivity

1.4. What is Machine Learning?

Machine learning is an algorithmic tool that is used to find patterns in historical data to make predictions on future data (Alpaydin, 2010). It is an automated process, that finds patterns and trends on its own based on the data presented to it and makes predictions on new data accordingly (Alpaydin, 2010). Over the years, machine learning has become its own field of study, with various applications and different ways to manipulate and use the data presented to it (Langley, 2011). As such, many different types of machine learning have surfaced based on the type of data presented to the algorithm and the type of predictions the algorithm is used to make (Langley, 2011). The two most distinct types of machine learning are supervised learning and unsupervised learning (Alpaydin, 2010). In supervised learning, the algorithm built is trained with both the input data and the output data. The input data would consist of the features or attributes that need to be analysed to make predictions, while the output data consists of the type of predictions that the algorithm needs to make (Alpaydin, 2010). The algorithm finds a connection between the input data and output data during the training stage, to then be able to make predictions given only input data (Alpaydin, 2010). In unsupervised learning, algorithms are not trained with desired output data. Instead, they use an iterative approach to analyse data and find connections to reach conclusions (Alpaydin, 2010).

Supervised learning can be divided into two main types, classification, and regression (Alpaydin, 2010). Classification is when the desired output is a class or a label. In that case, the algorithm is trained with already labeled data to be able to classify unlabelled data. Regression is when the desired output consists of one or more continuous variables (Alpaydin, 2010). An example of classification is using a model to determine a category of cars based on a set of features, while an example of regression could be the use of a model to determine the price of a car based on its mileage (Alpaydin, 2010) (Figure 1).

Figure 1- Classification plot (left) which identifies types of cars based on their price and engine power, and a regression curve (right) which identifies the correlation between a car's mileage and its price (Alpaydin, 2010).

1.5. Why Machine Learning?

Firstly, it easily identifies trends and patterns in data sets, even when those patterns are not apparent to humans (Langley, 2011). This is a very desirable advantage in this project, as the goal of the project is to produce a product that can allow a better understanding of sleep disorders. Being able to find and identify patterns between users' responses and the type of diagnosis that users receive may also allow reducing the number of questions that are to be asked for future users based on the most likely diagnoses they are likely to receive (Alpaydin, 2010).

Secondly, it allows for ongoing improvement (Langley, 2011). As machine learning models are used, the amount of data contained within them grows. With the expanding amount of information contained, a machine learning algorithm continues to identify patterns that may possibly lead algorithms to make more accurate predictions (Alpaydin, 2010). In the context of the questionnaire, this means that the algorithm may find shorter pathways to make a diagnosis based on patterns identified within users' responses, which in turn may help identify the key parameters and sleep habits that lead to certain sleep disorders.

1.6. Machine Learning Considerations

Although machine learning can be a powerful tool, it is important to be aware of its limitations when using it. Some of those limitations are listed below.

Firstly, the data needs to be representative of the general population. For example, if in the data used to build the model for this project, every person whose age is 35 also has sleep insomnia, the algorithm may make the link that every person whose age is 35 also has sleep insomnia, which is not necessarily true. This issue is referred to as overfitting (Dietterich, 1995; Hawkins, 2004). This means the machine learning algorithm is representative of the data it was trained with, but it does not accurately represent other data (Dietterich, 1995; Hawkins, 2004).

Secondly, time and resources. While a good machine learning model needs to have a large variety of data, it is also important to consider the increased computational power and increased time requirement that comes with the use of an increased amount of data to train the model (Alpaydin, 2010).

Thirdly, data representation (Alpaydin, 2010). To successfully meet the requirements of the machine learning model, it is important to represent the data in an appropriate manner before building the model (Alpaydin, 2010). The data needs to be presented in a manner that suits the application and appropriate for the machine learning algorithms to find patterns and trends without bias (Alpaydin, 2010).

1.7. Online Machine Learning

Online machine learning refers to a well-established learning paradigm with both theoretical and practical applications (Shalev-Shwartz, 2012). It allows the expansion of data used to construct the machine learning model used (Shalev-Shwartz, 2012). This is opposed to batch learning which only uses the samples used to construct the model to make predictions about future data (Shalev-Shwartz, 2012). Online learning would be extremely beneficial in this survey as it allows the model to continue learning as more data become accessible (Fontenla-Romero et al., 2013; Shalev-Shwartz, 2012). Therefore, allowing the model to keep adjusting to a growing dataset. While in this specific project online learning will not be explored as it adds other sources of complexity that simply cannot be accounted for due to time limitation. The goal of this project is to implement an online machine learning model that can continue to develop as more users answer the questionnaire.

2. Literature Review

2.1. Adaptive Questionnaires in Literature

Over the past two decades, there have been various projects that aimed to produce dynamically adaptive questionnaires (Kortum et al., 2017; Kurhila et al., 2001; Ortigosa et al., 2010). Those projects utilised various methods and techniques to personalise pre-existing questionnaires to users. Most of these projects made use of statistical analysis and information theory for the development of their algorithms. Other projects relied on the use of machine learning tools, such as decision trees, to develop their adaptive algorithm.

One of the early examples of adaptive questionnaires is the algorithm 'EDUFORM' which was developed in 2001 to help users identify their learning styles (Kurhila et al., 2001; Miettinen et al., 2005). The development process of the algorithm was composed of two phases. The first phase was referred to as the 'Profile creation phase' in which a set of characteristic profiles were captured in a model (Miettinen et al., 2005). The second stage was the 'Query phase', where the model was used for adaptive questioning (Miettinen et al., 2005). The profile creation phase involved constructing a model using responses provided by previous users to the same questionnaire (Miettinen et al., 2005). This was done using a Bayesian modeling approach, where probabilistic clusters from the itemised questionnaires were identified as classes (Kurhila et al., 2001; Miettinen et al., 2005).

The query phase involves identifying in which cluster users belong through adaptive questioning (Kurhila et al., 2001). A user's profile was represented by a probability distribution for the possible classification groups (Miettinen et al., 2005). As users answered the questionnaire, some groups became more likely than others (Miettinen et al., 2005). Users were presented with sets of 3-5 questions simultaneously. Based on users' responses, the set of unanswered questions with the highest information gain were the questions presented to users next (Miettinen et al., 2005). The algorithm determined the set of questions with the highest gain by calculating the Kullback-Leibler divergence between the current user profile and the profile that would be expected if answers to a specific set of questions were received (Miettinen et al., 2005). To carry out this method, the first set of questions asked was constant for all users, however, based on users' responses the following sets varied. Figure 2 shows an example of how users' responses were stored as users attempted the questionnaire. The first column identified the user, the second column showed the questions' identification numbers, and the remaining columns showed the probabilities of the possible answers. If a user had answered a question, one of the probabilities became 1 and the rest 0 (Miettinen et al., 2005).

Figure 2-The format used by Miettinen et al. to represent users' data during the process of answering their adaptive questionnaire. The first column is used to identify users and the second shows questions' numbers. The remaining columns represent the possible answer choices for a question. Rows, where a single value has the value '1' and the remaining values '0', indicate that the question had been answered by the user.

Ortigosa et al. (2010) also used pre-existing itemised questionnaires to build their adaptive algorithm (Ortigosa et al., 2010). Their dataset was described by the answers provided by students to the questionnaire (Ortigosa et al., 2010). Each question had two possible answers either a or b. The answers are used as attributes that describe each student to assign them to a class. Ortigosa et al. then used a C4.5 algorithm to build a decision tree as a classification tool with the aim of finding the most relevant question to ask (Ortigosa et al., 2010). The algorithm uses information entropy to build a decision tree using the training data provided (Ortigosa et al., 2010). The entropy in this context is described as the amount of information missing if a certain value is unknown (Ortigosa et al., 2010). Using this concept, the value of information gain provided by learning the answer to a question is calculated (Ortigosa et al., 2010). The algorithm then finds the question that provides the highest normalised information gain when answered to split the data (Ortigosa et al., 2010). The algorithm then recurs on smaller sub-lists and stops when all the samples in the list belong in the same class (Ortigosa et al., 2010). In that instance, the algorithm creates a leaf node. When none of the questions provide further information gain, a decision node is added to the tree using the expected tree (Ortigosa et al., 2010). The result of using this algorithm is a decision tree as shown in Figure 3 (Ortigosa et al., 2010).

Figure 3- A decision tree constructed by Ortigosa et al. to use for adaptive questioning. The tree was built using C4.5 algorithms and using the concept of information entropy. In the diagram, each occurrence of the letter 'Q' indicates a question. Other letters indicate possible user answers.

Kortum et al. (2017) aimed to produce a questionnaire for the diagnosis of rare diseases with dynamically channelized questions, such that patients would only need to answer questions relevant to their condition (Kortum et al., 2017). The questionnaire provides 53 questions about the symptoms suffered by patients and their perceived intensities. To build a model, 354 fully itemized questionnaires were used (Kortum et al., 2017). The questionnaires used in the model were all completed by patients who had been diagnosed by a medical doctor (Kortum et al., 2017). Their final diagnoses were used as the target data that classify each patient onto a separate group and their responses were used as the attributes that describe each patient (Kortum et al., 2017). Kortum et al. (2017) developed a system that is comprised of two modules, one is used for the selection of questions to ask and the second is used to make a diagnosis (Kortum et al., 2017). To select questions, each of the possible diagnoses is assigned a probability under the assumption that the user answers the remainder of questions in accordance with that diagnosis (Kortum et al., 2017). With the possibility that the user either answers consistently with one diagnosis or another, all the probabilities assigned to diagnoses are equal upon starting a new questionnaire (Kortum et al., 2017). For the questions that are yet to be asked, the answer provided most frequently for each of the possible diagnoses is assigned and a corresponding probability is computed (Kortum et al., 2017). Hence, odds are obtained for each diagnosis of an unanswered question (Kortum et al., 2017). For each of the odds, entropy is calculated, and the corresponding entropies are added (Kortum et al., 2017). The question that has the lowest sum of entropies allows for the highest information gain (Kortum et al., 2017). When a question is answered by the user, the user's answer replaces the corresponding response in the universal answer pattern of all possible diagnoses (Kortum et al., 2017). The diagnosis classification model is then used to compute the new corresponding probabilities for each of the possible diagnoses (Kortum et al., 2017). Four classifiers were used for the diagnosis module, Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Random Forest (RF) and Logistic Regression (LR) (Kortum et al., 2017). Each classifier is trained independently using the same raw data and each generates a separate result (Kortum et al., 2017). For the merging of results, a fusion algorithm is used. The algorithm derives a final diagnosis by evaluating each classifiers' compatibility and accuracy and makes a final prediction accordingly (Kortum et al., 2017).

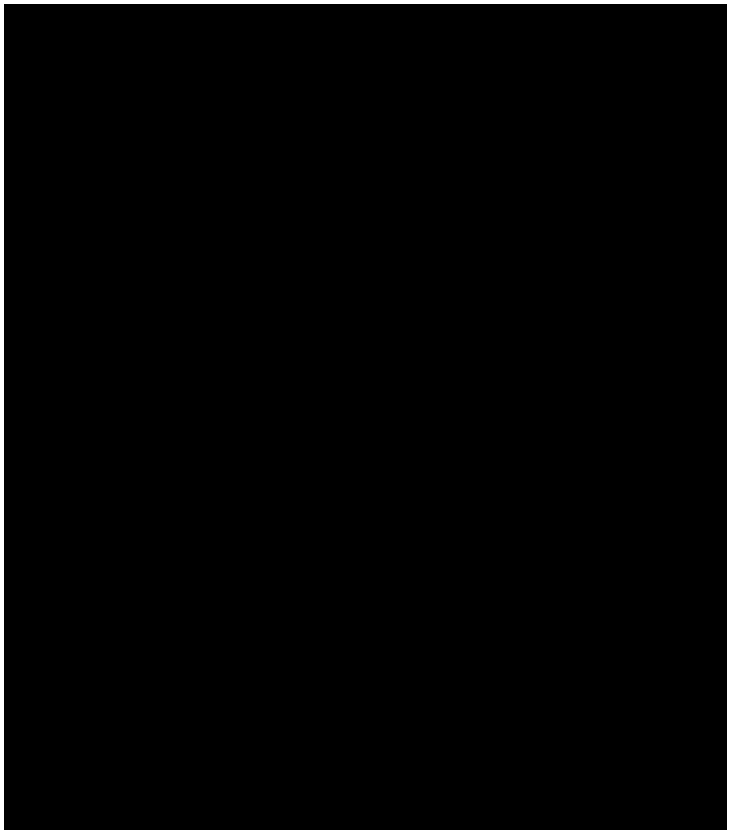

Figure 4- A hieratical structure showing the algorithm used by Kortum et al. to ask questions adaptively

2.2. Research Gap:

While various methods exist to construct adaptive questionnaires, all methods aim to reduce the number of questions through the calculation of information entropy. While those methods are effective, they do not provide the possibility of creating an online learning tool, their only purpose is to reduce the number of questions presented to users.

As such, the aim of this project is to create a simple method to create an adaptive questionnaire relying on machine learning tools. The desired outcome is one that can replicate the behavior of the existing question logic. Once the project is developed it can be paired with a diagnostic tool to allow for online learning to commence.

3. Methodology

3.1. The Existing Questionnaire

The existing questionnaire consists of over 100 questions each with a corresponding question ID in the form of a string. The questions are divided into six categories, SINGLECHOICE, CHECKBOX, EMAIL, PASSWORD, HIDDEN, and INT. SINGLECHOICE and CHECKBOX questions each accept a single response from an array of possible answers presented to users and only differ in the way in which they are presented. EMAIL and PASSWORD questions are self-descriptive, they accept a string for users' emails and passwords respectively. HIDDEN questions are prompts to the user, they only require the user to accept them to proceed. INT questions accept responses in the form of integers. Those questions involve entering times in which events occur (e.g. wake time) and are stored in seconds or questions about users' profiles, such as age, weight, height, partner's age, partner's height, and partner's weight. A question manifest file contains this information to describe each question based on their question ID, its contents are represented in Table 1.

Table 1- Structure of the question manifest file which contains all the questions, their question IDs, their category and their answer choices

Question ID	Description	Category	Responses	Units
email	Email Address	EMAIL		
password1	Password	PASSWORD		
password2	Confirm Password	PASSWORD		
beforeStartingSurvey	Please answer the following important questions before commencing the survey...	HIDDEN		
MedicalConditions	Are you currently under the care of a physician for depression and/or anxiety?	SINGLECHOICE	No, Yes	
Sex	What is your sex?	SINGLECHOICE	Female, Male, Intersex, Rather not say	
Age	What is your current age?	INT		Years

The existing questionnaire uses a logical static decision tree structure to decide which questions to present to users. In practice, IF/ELSE statements were used to build this logic. The aim of this project is to test the possibility of replacing this part of the code with a machine learning model that replicates the behavior of the IF/ELSE statements. The function that currently dictates which questions to ask is called 'calcNextQnID' and accepts two inputs. The first is the question ID which was last presented to the user in string format, and the second corresponds to user responses and can be a JavaScript Object Notation (JSON) object or a string. When the function is called, it returns a string of the question ID referring to the question that needs to be asked next. The function does not require both inputs to operate, only the responses input is needed for the function to return the correct value.

While responses can be inputted into the 'calcNextQnID' in both string or JSON object form, the function uses JSON objects to operate. As such, strings inputted into the function will be parsed to be in JSON form and need to be formatted accordingly. Ultimately, the function expects the responses input to contain the question IDs for the questions that had been asked, and the answers provided by a user to these questions. Table 2 illustrates the JSONs and strings that are accepted by the 'calcNextQnID' for a set of questions and the corresponding answers provided by a user to these questions. If the 'calcNextQnID' function was called with one of these inputs, the function would return a question ID in string format referring to the question that needs to be asked next.

Table 2 - Correct ways to format inputs to the 'calcNextQnID' logic which contains the question logic of the existing questionnaire

Question IDs for questions that had been asked	Answers provided by the user	Responses input to 'calcNextQnID' in string form	Responses input to 'calcNextQnID' in JSON form
email	user@email.com	"[{'email':'user@email.com', 'password1':'1234', 'password2':'1234',' MedicalCondition':'No', 'Sex':'Male', 'Age':'23'}]"	{'email':'user@email.com', 'password1': '1234', 'password2': '1234', 'MedicalCondition':'No', 'Sex': 'Male', 'Age': '23'}
password1	1234		
password2	1234		
MedicalCondition	No		
Sex	Male		
Age	23		

3.2. Existing Data

To build the machine learning model, responses provided by previous users are to be used. The responses are stored in a Comma-Separated Variables (CSV) file and are provided by 2,397 different users from various ages in the range 18 to 95. Respondents have varying working habits and are from varying occupational backgrounds. For each respondent, a corresponding user ID is included, and answers are stored under their associated question IDs. Table 3 illustrates the format in which data are stored in the CSV file.

Table 3- A table format illustration of the CSV file containing user data. The first column contains user IDs, the remaining columns each contain a question ID, and users' responses to that question

User ID	**Age**	**Sex**
1	23	Male
2	18	Male
3	47	Rather not Say
4	71	Female

It is important to note that the data was collected in an earlier version of the questionnaire, as such some of the question IDs and answers' formatting changed in the current version of the code.

However, the questions themselves and the number of possible responses to each question did not change.

3.3. Data Processing

To build a supervised machine learning model, data needs to be divided into features and target attributes (Alpaydin, 2010; Blum and Langley, 1997). Features are known parameters that describe a data point, target attributes are the parameters that are to be predicted (Alpaydin, 2010; Blum and Langley, 1997). To train a model, both the features and target attributes of data points are provided to the model (Alpaydin, 2010). The model can then be used to predict the target attributes of a data point using its features (Alpaydin, 2010). In this project, the target attributes to be predicted are the questions that need to be asked given a set of responses. The features are responses provided by users at a given point in the questionnaire. To build the machine learning model, the features need to be presented in a numerical format. The representation method needs to ensure that no sources of bias are introduced when converting data into a numerical form (Alpaydin, 2010). The bias would occur if features have unequal impacts on the predictions made by the model (Alpaydin, 2010). In this project, features were represented in arrays of zeros and ones. The process used to achieve this is described in this section.

3.3.1. Changes Made to Questionnaire

As mentioned in Section 3.2, the question IDs and the responses differ slightly in the questionnaire than in the collected user data. As the user data consists of data presented by almost 2,400 respondents, for over 100 questions, making changes to the formatting of user data would require a long time that cannot be accommodated by the time-constraints of this project. As such, a new question manifest file was developed to describe the older question IDs and their corresponding responses. A new version of the question logic was also developed to accept and return the old version of the question IDs.

As the project aims to develop a proof of concept, it was also decided to only include CHECKBOX and SINGLECHOICE questions in this version of the question logic. This decision will be discussed in detail in Section 3.3.2. The resulting question logic contained 84 questions from the original questionnaire and had seven branches.

3.3.2. Formatting of User Responses

To represent the data for the questionnaire's predictive algorithm, it was crucial to distinguish answered questions from unanswered questions at any given point. It was also important to achieve this without introducing sources of bias. For questions in SINGLECHOICE and CHECKBOX categories, a vector was constructed for each of the possible answers that can be chosen by a user. Each constructed vector consisted of ones and zeros, where the position of the one in the vector indicates the chosen answer. If a question was left unanswered or is yet to be answered a vector of zeros can be used to reflect that. The length of a vector reflects the number of possible answers to a given question. To further illustrate this, the question enquiring about the users' sex can be used as an example. A user can choose one of four possible answers, 'male', 'female', 'intersex', or 'rather not say'. A vector corresponding to each of these answers is constructed. Since there are four answers to choose from, the vectors used to describe the possible answers are each composed of four elements. If the question enquiring about the user's sex was left unanswered then a vector of four zeros is used to reflect that. Table 4 shows the vector representation for each of the possible answers for the question asking about users' sex. Using this method, users' answers can be represented in vector form when constructing the machine learning method.

Table 4 - The use of vectors to represent user responses. This is done so that the vectors would represent a users' features when creating a machine learning model

Answer	Corresponding Vector
Female	[1 0 0 0]
Male	[0 1 0 0]
Intersex	[0 0 1 0]
Rather Not Say	[0 0 0 1]
'NONE'	[0 0 0 0]

To implement this in code, the use of dictionaries was utilised. In object-oriented programming, a dictionary is an unordered data structure used to store a group of objects (McKinney, 2010). It contains a set of keys and their associated values. To access a value in a dictionary, its associated key has to be referred to. In this project, dictionaries had to be constructed in a way such that each answer choice for each question is associated with a corresponding vector that describes it. This meant that each of these answer choices had to be a key in a dictionary with its corresponding vector as the value associated with that key. Using the question manifest file, all the answer choices for every question could be accessed. However, since there are various questions, each with its corresponding answer choices, an individual dictionary had to be constructed for each question. Each of the dictionaries constructed had to be associated with their corresponding questions, as such, another dictionary was constructed to contain the dictionaries constructed for individual questions. The process of containing dictionaries inside a dictionary is referred to as 'Nesting' (Chugh, 2013). The term 'Nested Dictionary' refers to a dictionary that is inside another dictionary. In this project, the nested dictionaries are the dictionaries that contain answer choices as values and the question IDs corresponding to these dictionaries as keys. The structure of the dictionary used is illustrated in Figure 5.

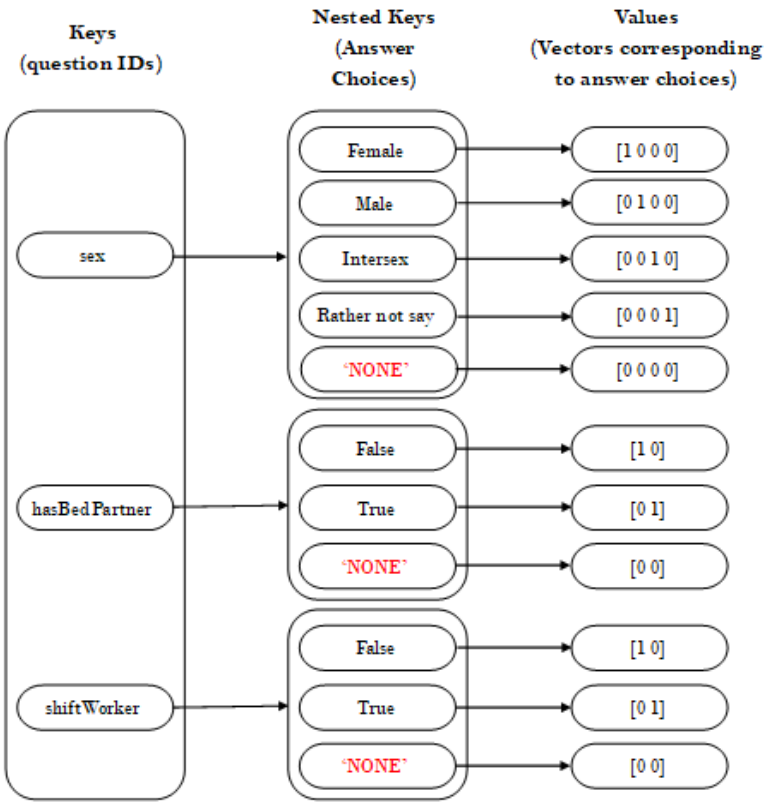

Figure 5 – The use of nested dictionaries to store questions, answer choices to each question and vector forms of those answers.

To build the parent dictionary, a FOR loop was used that ran through the contents of the question manifest file. For each of the questions that belonged in SINGLECHOICE or CHECKBOX categories, a dictionary was created to contain the possible answer choices and their corresponding vectors. At the end of each iteration of the FOR loop, the question ID and the constructed vector were stored as the key and value of the dictionary respectively. Figure 6 shows a flowchart that represents the way the FOR loop operated. Once the FOR loop had iterated over all the contents of the question manifest file, the resulting parent dictionary contained all the question IDs for SINGLECHOICE and CHECKBOX questions and a corresponding nested dictionary that contained all the possible answer choices for each of those questions.

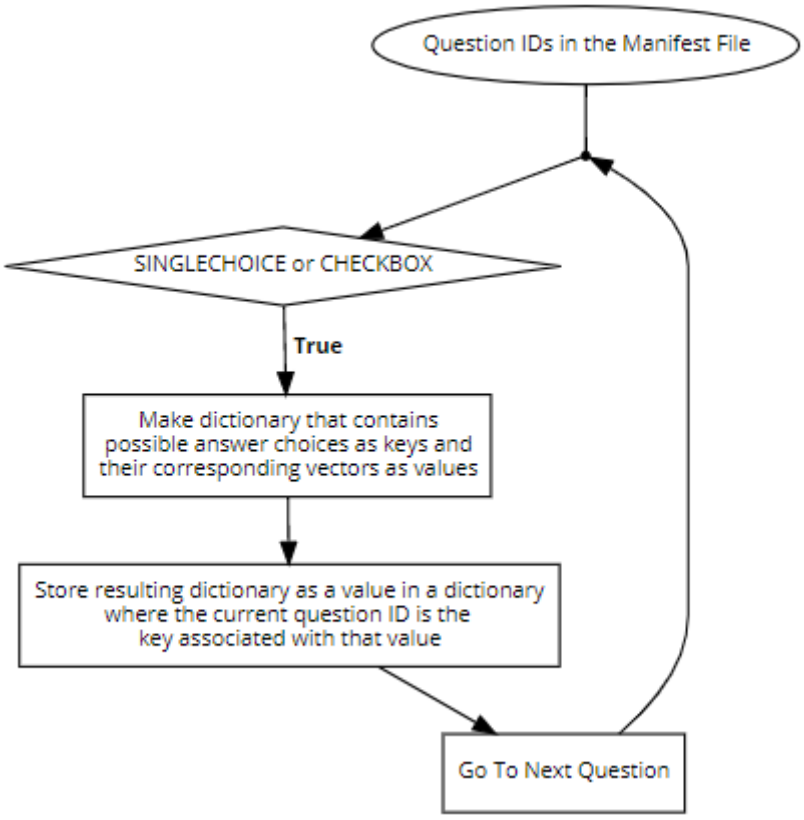

Figure 6- Flow diagram showing the process of creating nested dictionaries to use in the generation of features.

To build the nested dictionaries, a FOR loop was contained inside the FOR loop used to build the parent dictionary. The inner FOR loop iterated over the possible answer choices of a given question. In each iteration, a vector of length equivalent to the number of possible answer choices for a given question was created. All the elements inside that vector were set to have a value of '0' except for the element in the position corresponding to the number of iterations the FOR loop had passed. That element is assigned a value of '1' instead. An IF statement was added to check whether a 'NONE' key existed in the nested dictionary or not. In case it did not, a 'NONE' key was created with a vector of zeros assigned as its value. Figure 7 shows a flowchart that represents the operation of the FOR loop used to build the nested dictionaries.

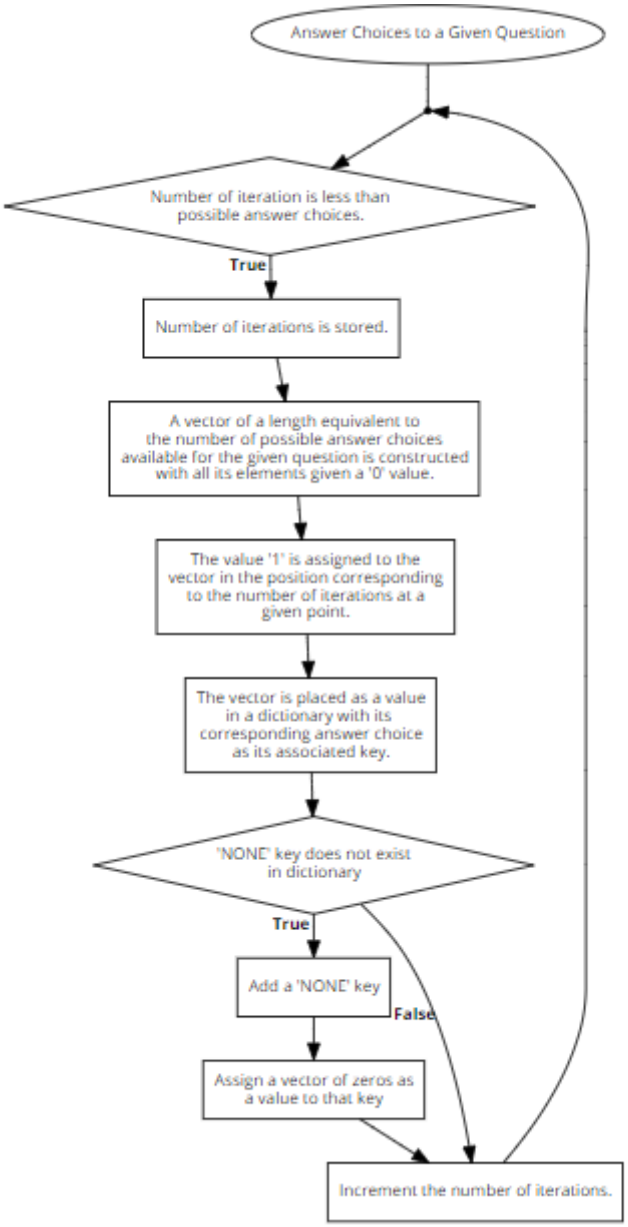

Figure 7 - Flow diagram showing the process used to create vectors corresponding to answer choices for each question

Using the parent dictionary provided an easily accessible tool that can be used to represent user responses to any SINGLECHOICE or CHECKBOX question. Four more question categories were present in the questionnaire. EMAIL, PASSWORD and HIDDEN category questions were not present in the complete user data CSV file that would be later used for creating a machine learning model. This led to the decision of removing the questions that belonged in these categories from the question logic. Answers to INT questions were already in numerical format. However, using them in their original form would not allow representing unanswered questions. As mentioned previously, in this machine learning approach it is crucial to represent both answered and unanswered questions. To illustrate this issue, the question enquiring about users' weight can be used as an example. Not every user was required to enter their weight in the questionnaire. If every occurrence of unanswered weight question in the data was represented using a zero value, this would cause the distribution of weight data to have a positively skewed curve at the value of zero, while none of the actual users could possibly weigh zero kilograms. Thus, a source of bias would be introduced to the data.

A different method had to be utilised for the representation of the numerical answers provided to the questions in the INT category. But due to time-constraints, it was decided to remove INT questions from the question logic. Chapter 6 contains a suggested method for the representation of answers to the questions that fall under the INT category.

3.3.3. Formatting of Questions

As stated previously, the target attributes for the predictive models in the project are the question IDs that correspond to the questions that need to be asked given a set of responses. While the use of the existing question IDs in their string format is adequate to be used as target attributes, a decision was made to represent the questions in two different forms in this project. This was done to test the effect that different was of formatting target attributes have on the accuracy of the predictive algorithm. The first form used to represent the questions was the existing question IDs as strings. The second form used was in vector format. The vectors generated for the questions were each composed of 84 elements, equivalent to the 84 questions that were contained in the altered question logic. To associate the new vectors with their question IDs, a new dictionary was constructed. Each vector had a single element with the value '1' while the remaining elements were zeros. The position of the '1' in the vector represented the question's order. Figure 8 shows a representation of how questions were represented in vector form.

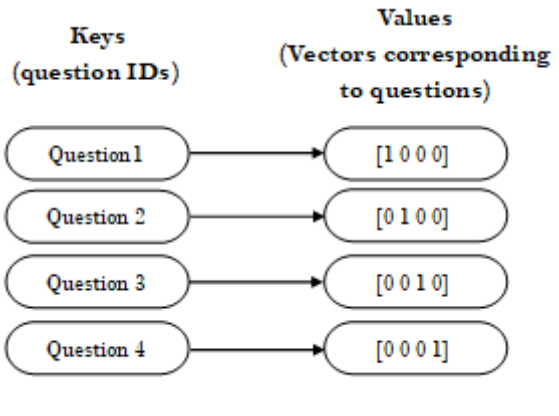

Figure 8- The use of dictionaries to represent each question with a single vector. The length of each of the vectors reflects the length of the questionnaire. Since the example contains vectors which are four-elements-long, that indicates that the full questionnaire in the example is composed of 4 questions.

3.3.4. Generation of Features and Target Attributes from Raw Data

After developing methods that allowed the conversion of data from their original form to a form that can be used to build a machine learning model. Those methods had to be applied to the user data contained in the CSV file. Since the aim of the project is to build a model that can learn and replicate the existing question logic of the questionnaire, data representation had to reflect that logic. Since user data in the CSV file contained complete questionnaires filled by users, the question logic had to be applied to the data in the file prior to modeling. In other words, the data provided by users had to be used to complete the questionnaire using the question logic. Throughout the process of completing the questionnaire, the features and target attributes had to be stored. The questionnaire had to be completed separately by each individual user who would be distinguished by their unique user ID. The format of data that would be used to build the predictive model had to contain users' IDs, the responses provided by users to questions at given points of the questionnaire, and the questions that need to be presented to users given the responses they had provided. Table 5 shows the form in which data had to be represented prior to building predictive models.

Table 5 -Desired representation of data prior to building machine learning models. Target attributes the question to be asked and they are represented in two ways as strings and as vectors. The table shows an example where the full questionnaire is composed of 4 questions. Notice that each users' responses are recorded iteratively until the questionnaire is complete. The table shows an example where only one of the questionThree or questionFour needs to be answered.

User ID	Features	Target Attributes (Vector Form)	Target Attributes (String Form)
1	[0 0 0 0 0 0 0 0]	[1 0 0 0]	questionOne
1	[0 1 0 0 0 0 0 0]	[0 1 0 0]	questionTwo
1	[0 1 0 0 1 0 0 0]	[0 0 1 0]	questionThree
2	[0 0 0 0 0 0 0 0]	[1 0 0 0]	questionOne
2	[1 0 0 0 0 0 0 0]	[0 1 0 0]	questionTwo
2	[1 0 0 0 0 1 0 0]	[0 0 0 1]	questionFour
…	…	…	…

As mentioned previously, each row in the CSV file contains the responses provided by a user. So, to complete the questionnaire using user data in code, a FOR loop was used that iterated through the contents of that file. In every iteration of the loop, an empty JSON object was created. Then inside a WHILE loop, the question logic function 'calcNextQnID' was used to choose what questions to ask based on the contents of the JSON object. If the JSON object was empty, the first question in the questionnaire was asked. The responses inside the JSON file were then converted into their vector form, which would be a vector of zeros in the case of an empty JSON. The question ID determined by the 'calcNextQnID' function was also converted into its vector form. Both vectors obtained were stored alongside the user ID, and the question ID determined by the 'calcNextQnID' into a 'Pandas Dataframe', which is a two dimensional, size-mutable data structure (McKinney, 2012). The WHILE loop continued to run until the questionnaire was completed by the contents of a users' data. Each iteration of the WHILE loop added a row to the data frame, where each row contained a user's ID, the responses provided by that user in the form of a single vector, the question that was asked by the question logic based on the user responses in both vector and string forms. The operation of the code used to construct the data frame is represented by a flowchart in Figure 9.

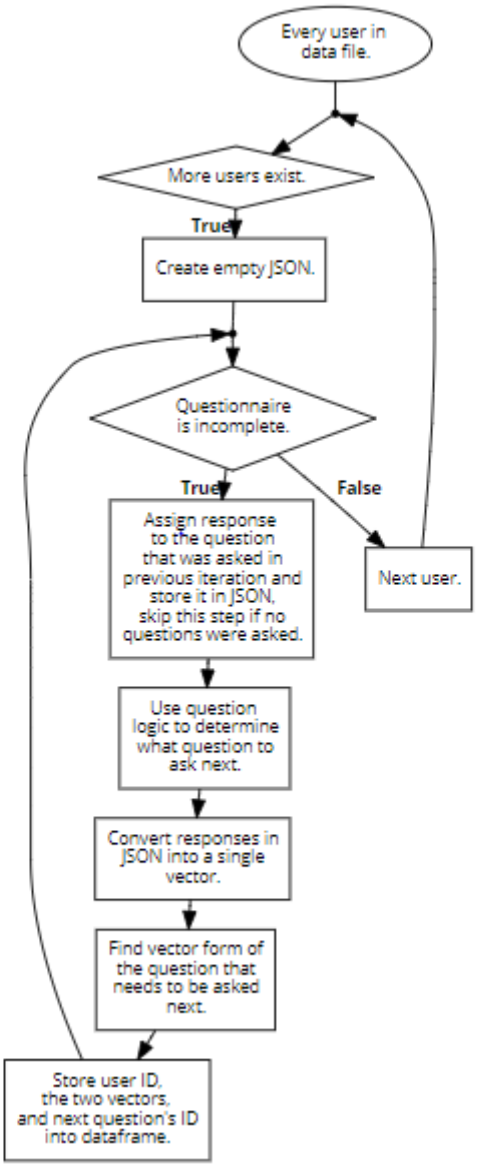

Figure 9 - Flow diagram representing the method used to construct a data frame. The contents of the data frame would later be used for the creation of machine learning models

Due to memory limitations, it was not possible to store all the data for every single user in the CSV file. The method mentioned previously stored user data at every point of the questionnaire. That meant the number of rows added to the data frame for a single user would have been equal to the number of questions that the user had answered. Each row in the data frame would contain long vectors to represent the questions and responses which could not be fully contained in the data frame. As such, two methods were employed to store variables into the data frame. Both methods replicated the operation of the method introduced previously, however, conditions were added prior to the storage phase.

The first method of storing data operated exactly like the previously introduced method, however, instead of applying the question logic to every single user in the CSV file, it was only applied to 500 randomly selected users based on their user ID and store their responses in the data frame as discussed previously. The resulting data frame had the same structure as shown in Table 5.

The second method was to apply the question logic to every single user in the CSV file, but only randomly store a portion of the data in a data frame. This means that the same mechanism for building the data frame would be employed, however, instead of storing the responses and questions to be asked at every single point of the questionnaire, storing data was only done at randomly selected points of the questionnaire. Using this method meant that the data frame would contain data from all users, but not at every single point of the questionnaire. The resulting data frame from this method would have the structure shown in Table 6, by comparing Table 5 to Table 6 the difference between the two methods can be illustrated. Since there are 84 questions in the questionnaire used, a one-in-eight chance was used to decide the stages of the questionnaire where user responses were stored in the data frame. However, since not all questions were asked to all users, it meant that on average more than 10% of each users' data was included in the resulting data frame of this method.

Table 6 – Method 2 used to store data, notice that not all questions asked to every user are stored in the data frame. The algorithm decides which parts of the data are stored randomly.

User ID	Features	Target Attributes (Vector Form)	Target Attributes (String Form)
1	[0 0 0 0 0 0 0 0]	[1 0 0 0]	questionOne
1	[0 1 0 0 0 0 0 0]	[0 1 0 0]	questionTwo
2	[1 0 0 0 0 1 0 0]	[0 0 1 0]	questionFour
3	[0 1 0 0 0 0 0 0]	[1 0 0 0]	questionTwo
4	[1 0 0 0 0 0 0 0]	[0 1 0 0]	questionTwo
4	[1 0 0 0 0 1 0 0]	[0 0 0 1]	questionFour
…	…	…	…

Both employed methods provided advantages and disadvantages, the first method's advantage was that it guaranteed that data was stored at every single point of the questionnaire for the users who had their data included in the model. However, the disadvantage it brought was that it contained a portion of the users whose data are available which reduced the number of variations contained in the data frame. The second method employed provided the advantage of providing more variation in data as it contained more users. However, it also meant that only a portion of users' data was included in the resulting models.

The use of the second method for creating data frames provided an advantage in the distribution of data. In the case in which the first method was used, for every user, all the vectors would contain a users' response to the first question, except the vector generated at the start of the questionnaire in which no questions had been answered yet. That meant responses for questions at the start of the questionnaire by a specific user appeared recurrently in the data. The use of the second feature reduced that effect, as not every response was stored for every user. This meant that a specific user's responses for early parts of the questionnaire were less recurrent in the data frame when the second method was used. Dataframes created using both methods were used in the production of predictive models, this was to compare both methods based on the accuracy of the models they were used to produce.

3.4. Predictive Modelling

Various models were tested in their ability to replicate the question logic. 'Scikit-learn' tools were used to build the algorithm for each of the predictive models constructed. Scikit-learn is a Python library that provides simple and efficient tools for data mining and machine learning (Pedregosa et al., 2011). The models tested for the vector output format all had to support multilabel predictions (Garreta and Moncecchi, 2013; Hackeling, 2017). As such, the models used were 'Neural Network', 'Decision Tree', 'Random Forest' and 'K-Nearest Neighbours' models. For string output format, 'Support Vector Machine' and 'Naïve Bays' models were also tested.

3.4.1. Neural Network Modelling

Neural networks are composed of an interconnected group of nodes. The leftmost layer of nodes is the input layer consisting of the features (Haykins, 2009). The input layer is followed by hidden layers which consist of weightings that are refined over rune as the network is trained (Haykins, 2009; Krogh, 2008). Neurons in hidden layers transform values from previous layers with weighted linear summation followed by a non-linear activation function (Haykins, 2009; Krogh, 2008). The rightmost layer of the network is the output layer which transforms the values from the last hidden layers into output values (Haykins, 2009; Krogh, 2008).

Figure 10 - A diagram representing the structure of a neural network. The network is composed of input, output and a single hidden layer (" Control Systems Technology Group," n.d.).

The neural network algorithm built using Scikit-learn trains using a gradient descent, where the gradients are calculated using Backpropagation(Garreta and Moncecchi, 2013; Hackeling, 2017). The algorithm then minimises the cross-entropy loss function to make predictions (Garreta and Moncecchi, 2013; Hackeling, 2017).

The Scikit-learn function used to build neural networks accepts several variables. The parameters relevant to this project were the following:

- Number of hidden layers, which was set to one
- Size of hidden layers, which was set to 173. This number was determined using the geometric pyramid rule (Cheng and Adams, 1995)
- The solver used for weight optimisation, which was set to the Limited-memory Broyden–Fletcher–Goldfarb–Shanno optimiser.
- Learning rate schedule for weight updates which was set to constant.

3.4.2. Decision Tree Modelling

Decision trees classify instances by sorting them based on features value (Kotsiantis, 2007). The nodes in decision trees represent the features that are to be classified (Kotsiantis, 2007). The branches of the decision trees represent values that can be assumed(Kotsiantis, 2007). The model makes predictions by learning simple Boolean rules inferred from the features (Kotsiantis, 2007). A decision tree was created using Scikit-learn by calling the function 'DecisionTreeClassifier()', then fitting the desired data into it (Pedregosa et al., 2011).

Figure 11- Structure of a decision tree (DataCamp, n.d.)

3.4.3. Random Forest Modelling

Random forest uses ensemble methods to make predictions. A random forest consists of a series of decision trees. Each tree makes a prediction based on features presented to them. The prediction that has the highest number of votes is the output of the random forest model. Using Scikit learn, random forest classifiers containing 50 decision trees were constructed.

Figure 12 - Structure of a random forest predictive model (Koehrsen, 2017)

3.4.4. K-Nearest Neighbours Algorithm

K- The nearest neighbour algorithm makes predictions by finding the features closest to an instance (Deng et al., 2016). Those features are referred to as neighbours (Alpaydin, 2010; Deng et al., 2016). The number of neighbours per instance can be determined in is referred to as the K value. An instance is given a class based on the classes of points that have the closest features to that instance (Deng et al., 2016). To find similar points, distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance (Alpaydin, 2010; Kotsiantis, 2007). In this project, the number of neighbours was set to three, and distance was calculated using Minkowski distance. The number of neighbours was chosen to be 3 as odd numbers give the advantage of having a vote. When an instance's neighbours are identified, a prediction is made based on the target attributes of most of the neighbours. The number of neighbours was set to be low, as features need to be as closely related their neighbours as possible.

Figure 13 - Representation of k-nearest neighbours

3.4.5. Support Vector Machine Model

A support vector machine model uses hyperplanes to separate data. Once a support vector had been trained, it is able to output an optimal hyperplane to separate new data into their categorical differences. That way, a support vector machine can separate features based on their corresponding target attributes, and predictions would be made accordingly.

Figure 14 - Operation of support vector machine models (Talpur, 2017)

3.4.6. Naive Bayes Model

Naïve Bayes model uses Bayes theorem (Equation 1) to make predictions about a set of features with the 'naïve' assumption that every pair of features given a target value are conditionally independent (Zhang, 2004).

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Equation 1 - Bayes theorem

Equation 2 is obtained when the conditional independence rule is applied to Equation 1 (Zhang, 2004).

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

Equation 2 - Bayes theorem after applying the naive conditional independence

Equation 2 is then simplified to Equation 3 for all i (Zhang, 2004).

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Equation 3 – Simplified Bayes theorem

The classification rule in Equation 4 is then used (Zhang, 2004).

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

Equation 4 - Classification rule application to Bayes theorem

Finally, Maximum A Posteriori (MAP) is used to estimate the relative frequency of the target data in the training set $(P(y))$ and the probability that an instance corresponds to a given target attribute $(P(x_i|y))$ (Hackeling, 2017; Zhang, 2004). Different naïve Bayes model differ by the assumptions they make regarding the distribution of $P(x_i|y)$ (Hackeling, 2017). In this project, 'Gaussian Naive Bayes' which implements the Gaussian Naive Bayes algorithm to make predictions. The likelihood of the features is assumed to be Gaussian (Equation 5) (Garreta and Moncecchi, 2013).

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Equation 5 – Gaussian/ Normal distribution

3.5. Testing Methods

The next step after creating machine learning models was to test their ability to replicate the behavior of the question logic in predicting questions to ask based on user responses. 'Cross-Validation' was used to do this. In addition, confusion matrices were constructed for each of the tests that were run.

3.5.1. Cross-Validation

To test the resulting model ten-fold cross-validation was performed on the data. That was done, by splitting the data into 10 groups, 9 of the groups were used to train the model and the remaining group to test it. The process was then repeated but the group used to test the data previously was added into the training set instead, and a group from the training set was used to test the model. The process is repeated ten times, each time with a different testing set, this process is depicted in Figure 16. Grouping data in this project was done using the user IDs of respondents. That is to ensure that no bias is introduced by potentially testing the model with responses from a user whose data is included in the training set.

The use of this method ensures that the resulting models are not biased. It validates that the model constructed is not representative of the data it was trained with but can be used for other data. After performing each test, an accuracy score was obtained by comparing predicted data to true data. Confusion matrices were also constructed at the end of each fold to understand sources of inaccuracies.

Figure 15 - Representation of 10-fold cross-validation test (Talpur, 2017)

3.5.2. Confusion Matrices

A confusion matrix is an array that represents the predictive performance of a model. It is a square matrix that has a number of rows and columns equivalent to the number of possible predictions that can be made by the model. The matrix shows a record of the number of times the predictive model made a correct prediction, and the number of times it failed to do so. Confusion matrices could also be normalised to show ratios of accurate and inaccurate predictions.

Since the vector format used to represent the target data contained 84 elements, that meant that creating confusion matrices in that form would produce confusion matrices that have 84 dimensions, which would not be usable. As such confusion matrices were only produced for models that have their target attributes in the form of a string.

Figure 16 - Representation of a confusion matrix (Narkhede, 2019)

4. Results

As stated in Chapter 3, two methods of generating features were used. The first involved storing all users' responses for each of the 500 users, this will be referred to as 'Method 1'. The second involved storing data from every user's questionnaire at random stages of the questionnaire, this method will be referred to as Method 2. Both methods were employed to both predict target attributes in string and vector forms. The results are displayed in this chapter.

4.1. Neural Network Models

4.1.1. Method 1 With Target Attributes in String Form

Table 7 shows the accuracy scores for cross-validation tests performed on a neural network model trained with data stored using method 1 and target attributes in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 4.1%. This indicates that the model was consistent, and it confirms that the model does not suffer from issues relating to overfitting. On average, the predictive model had an error rate of 18.8%.

Table 7 - Cross-validation results for neural network models trained with data stored using method 1 and target data in string form

Fold #	Accuracy (%)
1	84.1
2	84.5
3	80.4
4	81.4
5	82.5
6	80.7
7	83.5
8	80.8
9	82.8
10	81.7
Average	82.2

Figure 17 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to predict accurately. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions towards the end of the questionnaire also had relatively high error rates.

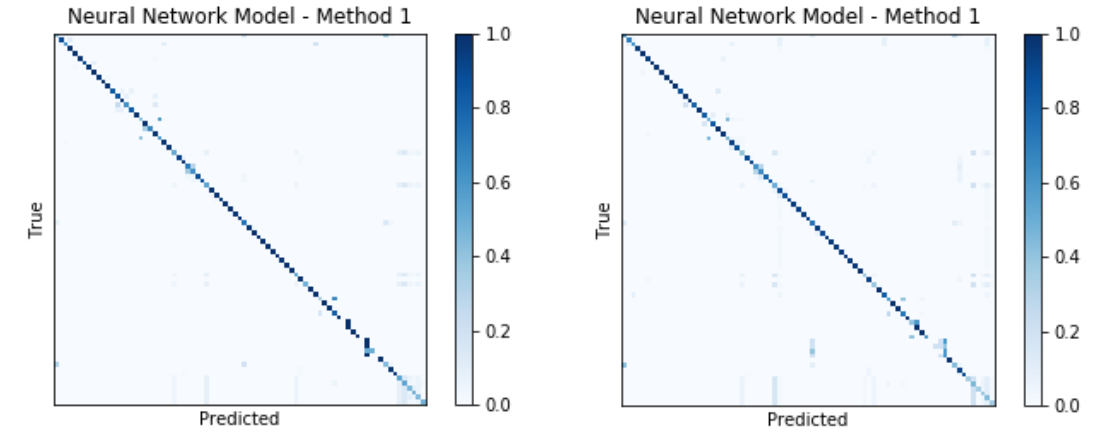

Figure 17 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a neural network model trained with data stored using method 1 and target attributes in string form.

4.1.2. Method 2 With Target Attributes in String Form

Table 8 shows the accuracy scores for cross-validation tests performed on a neural network model trained with data stored using method 2 and target attributes in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 3.5%. This indicates that the model was consistent, and it confirms that the model does not suffer from issues relating to overfitting. On average, the predictive model had an error rate of 21.2%.

Table 8-Cross-validation results for neural network models trained with data stored using method 2 and target data in string form

Fold #	Accuracy (%)
1	79.8
2	79.2
3	79.4
4	79.1
5	79.4
6	78.1
7	79.4
8	81.6
9	80.4
10	82
Average	79.8

Figure 18 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to predict accurately. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions towards the end of the questionnaire also had relatively high error rates.

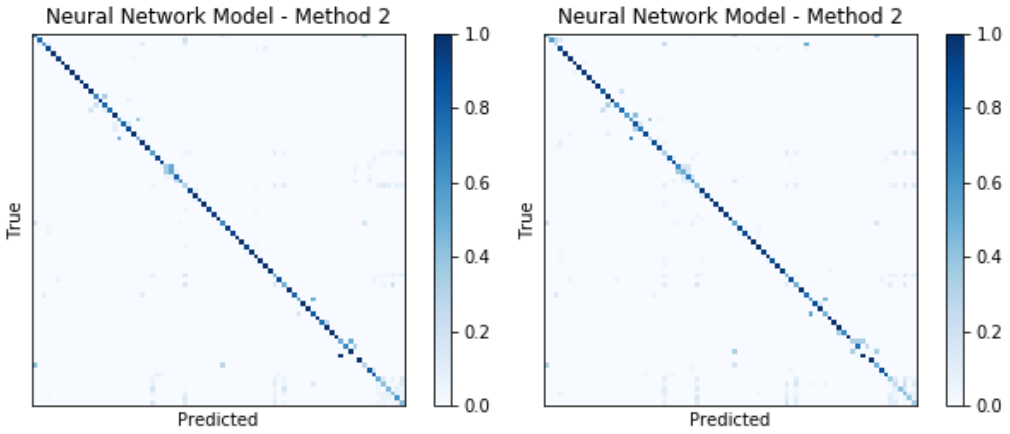

Figure 18 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a neural network model trained with data stored using method 2 and target attributes in string form.

4.1.3. Method 1 With Target Attributes in Vector Form

Table 9 shows the accuracy scores for the cross-validation tests performed on a neural network model trained with data stored using method 1 and target attributes in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 10%. This indicates that the model was inconsistent. Different groups led to variations in the model's ability to make predictions. On average, the predictive model had an error rate of 30.9%.

Table 9 - Cross-validation results for neural network models trained with data stored using method 1 and target data in vector form

Fold #	Accuracy (%)
1	62.7
2	66.9
3	72.7
4	69.5
5	71.5
6	72
7	67.3
8	69.1
9	69.6
10	69.6
Average	69.1

4.1.4. Method 2 With Target Attributes in Vector Form

Table 10 shows the accuracy scores for cross-validation tests performed on a neural network model trained with data stored using method 2 and target attributes in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 5.7%. This indicates that the model was inconsistent. Different groups lead to variations in the model's ability to make predictions. On average, the predictive model had an error rate of 33.9%.

Table 10-Cross-validation results for neural network models trained with data stored using method 2 and target data in vector form

Fold #	Accuracy (%)
1	62.6
2	64.3
3	68
4	67.9
5	67
6	65.7
7	67.2
8	68.3
9	65.2
10	64.4
Average	66.1

4.2. Decision Tree Models

4.2.1. Method 1 With Target Attributes in String Form

Table 11 shows the accuracy scores for cross-validation tests performed on a decision tree model trained with data stored using method 1 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 7.2%. This indicates that the model was consistent. On average, the predictive model had an error rate of 25.2%.

Table 11 - Cross-validation results for decision tree models trained with data stored using method 1 and target data in string form

Fold #	Accuracy (%)
1	77
2	77.9
3	74
4	73.5
5	74.8
6	70.7
7	77.6
8	74.2
9	76
10	72.6
Average	74.8

Figure 19 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions towards the end of the questionnaire also had relatively high error rates.

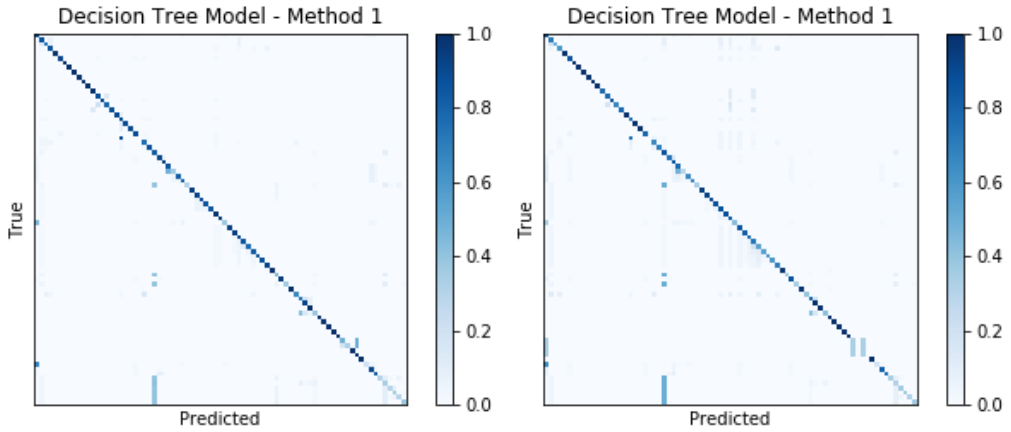

Figure 19 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a decision tree model trained with data stored using method 1 and target attributes in string form

4.2.2. Method 2 With Target Attributes in String Form

Table 12 shows the accuracy scores for cross-validation tests performed on a decision tree model trained with data stored using method 2 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 4.6%. This indicates that the model was consistent. On average, the predictive model had an error rate of 30.8%.

Table 12 - Cross-validation results for decision tree models trained with data stored using method 2 and target data in string form

Fold #	Accuracy (%)
1	66.9
2	69.8
3	70.2
4	70.1
5	69.3
6	69.2
7	68.2
8	68
9	69.3
10	71.3
Average	69.2

Figure 20 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions towards the end of the questionnaire also had relatively high error rates.

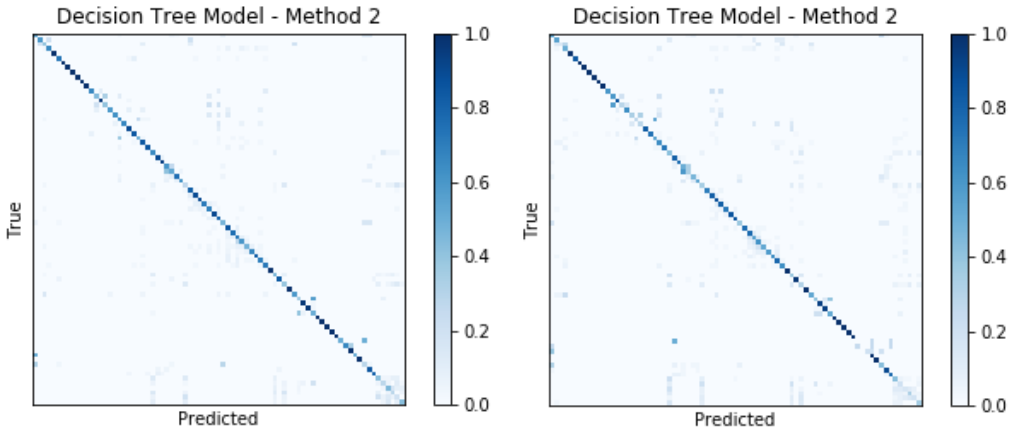

Figure 20 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a decision tree model trained with data stored using method 2 and target attributes in string form

4.2.3. Method 1 With Target Attributes in Vector Form

Table 13 shows the accuracy scores for cross-validation tests performed on a decision tree model trained with data stored using method 1 and target attributes being in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 6.5%. This indicates that the model was consistent. On average, the predictive model had an error rate of 30.3%.

Table 13 - Cross-validation results for decision tree models trained with data stored using method 1 and target data in vector form

Fold #	Accuracy (%)
1	65.1
2	67.3
3	70.8
4	69.4
5	71.1
6	71
7	71.8
8	70.4
9	68.2
10	71.6
Average	69.7

4.2.4. Method 2 With Target Attributes in Vector Form

Table 14 shows the accuracy scores for cross-validation tests performed on a decision tree model trained with data stored using method 2 and target attributes being in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 6.4%. This indicates that the model was consistent. On average, the predictive model had an error rate of 32%.

Table 14- Cross-validation results for decision tree models trained with data stored using method 2 and target data in vector form

Fold #	Accuracy (%)
1	64.4
2	67.5
3	69.4
4	68.4
5	68.2
6	70.8
7	67.8
8	69
9	67.2
10	67.7
Average	68

4.3. Random Forest Models

4.3.1. Method 1 With Target Attributes in String Form

Table 15 shows the accuracy scores for cross-validation tests performed on a random forest model trained with data stored using method 1 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 5.4%. This indicates that the model was consistent. On average, the predictive model had an error rate of 29.2%.

Table 15 - Cross-validation results for random forest models trained with data stored using method 1 and target data in string form

Fold #	Accuracy (%)
1	73
2	72.5
3	70.4
4	69.5
5	70.4
6	67.6
7	72
8	69.4
9	72.5
10	70.4
Average	70.8

Figure 21 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions towards the end of the questionnaire also had relatively high error rates.

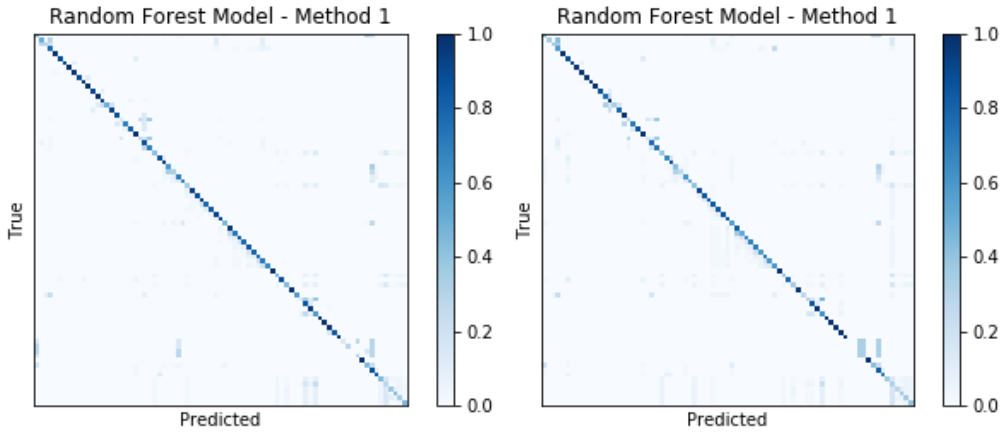

Figure 21 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a random forest model trained with data stored using method 1 and target attributes in string form

4.3.2. Method 2 With Target Attributes in String Form

Table 16 shows the accuracy scores for cross-validation tests performed on a random forest model trained with data stored using method 2 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 2.9%. This indicates that the model was consistent. On average, the predictive model had an error rate of 36.1%.

Table 16 - Cross-validation results for random forest models trained with data stored using method 2 and target data in string form

Fold #	Accuracy (%)
1	63.9
2	64.6
3	64.8
4	63.6
5	63.6
6	64.4
7	63
8	62.4
9	63.4
10	65.3
Average	63.9

Figure 22 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions at the beginning and end of the questionnaire also had relatively high error rates.

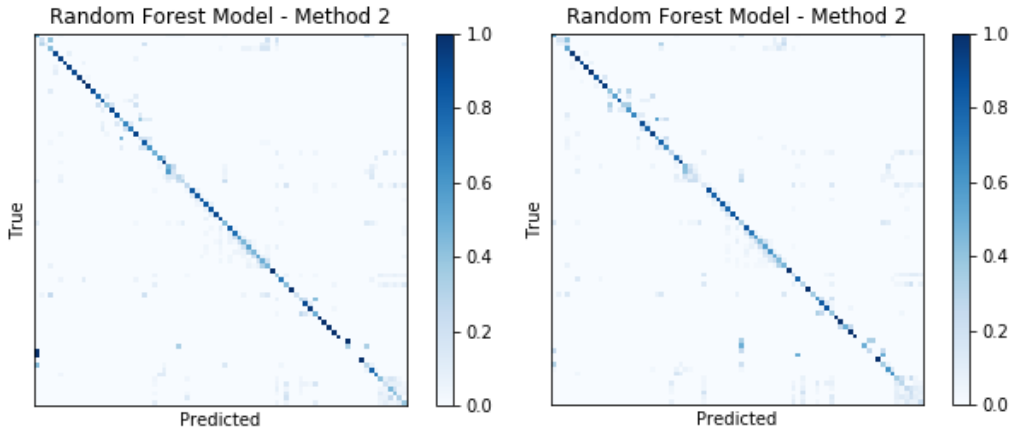

Figure 22 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a random forest model trained with data stored using method 2 and target attributes in string form

4.3.3. Method 1 With Target Attributes in Vector Form

Table 17 shows the accuracy scores for cross-validation tests performed on a random forest model trained with data stored using method 1 and target attributes being in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 5.3%. This indicates that the model was consistent. On average, the predictive model had an error rate of 60.2%.

Table 17 - Cross-validation results for random forest models trained with data stored using method 1 and target data in vector form

Fold #	Accuracy (%)
1	36.4
2	39.8
3	41.7
4	40.6
5	41.3
6	40.7
7	41.1
8	39
9	39.1
10	38.1
Average	39.8

4.3.4. Method 2 With Target Attributes in Vector Form

Table 18 shows the accuracy scores for cross-validation tests performed on a random forest model trained with data stored using method 2 and target attributes being in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 5.3%. This indicates that the model was consistent. On average, the predictive model had an error rate of 60.2%.

Table 18 - Cross-validation results for random forest models trained with data stored using method 2 and target data in vector form

Fold #	Accuracy (%)
1	18.6
2	19.5
3	20.5
4	19.4
5	19.1
6	19.2
7	17.7
8	17.1
9	19.3
10	17.5
Average	18.8

4.4. K-Nearest Neighbour Models

4.4.1. Method 1 With Target Attributes in String Form

Table 19 shows the accuracy scores for cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 1 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 2.8%. This indicates that the model was consistent. On average, the predictive model had an error rate of 79.1%.

Table 19 - Cross-validation results for k-nearest neighbour models trained with data stored using method 1 and target data in string form

Fold #	Accuracy (%)
1	19.5
2	19.8
3	21.5
4	22.3
5	21
6	20.3
7	21.6
8	21.7
9	20.7
10	20.6
Average	20.9

Figure 23 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. However, they show scattered results and no clear trend on why the errors occurred. This indicates that the modeling technique was not sufficient to obtain reliable results.

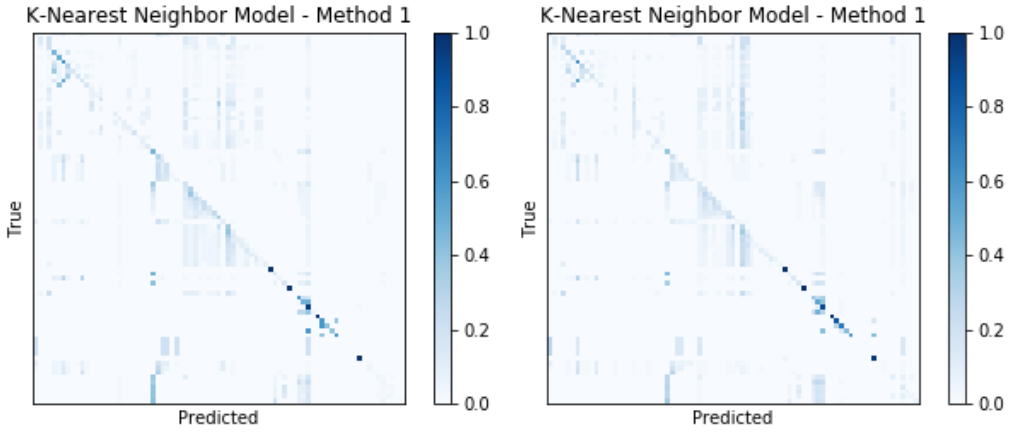

Figure 23 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 1 and target attributes in string form

4.4.2. Method 2 With Target Attributes in String Form

Table 20 shows the accuracy scores for cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 2 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 2.5%. This indicates that the model was consistent. On average, the predictive model had an error rate of 82.6%.

Table 20 - Cross-validation results for k-nearest neighbour models trained with data stored using method 2 and target data in string form

Fold #	Accuracy (%)
1	16.9
2	19.1
3	17.6
4	17.5
5	17
6	17.9
7	16.8
8	16.9
9	16.6
10	17.7
Average	17.4

Figure 24 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. However, they show scattered results and no clear trend on why the errors occurred. This indicates that the modeling technique was not sufficient to obtain reliable results.

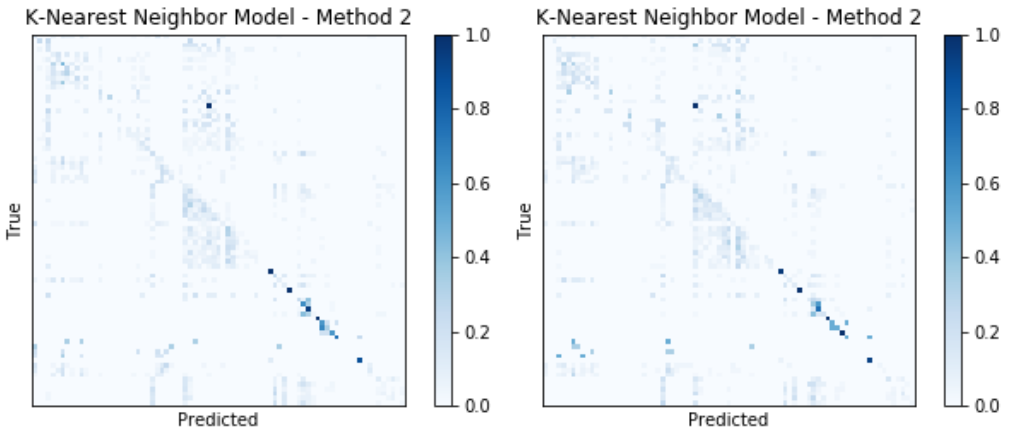

Figure 24 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 2 and target attributes in string form

4.4.3. Method 1 With Target Attributes in Vector Form

Table 21 shows the accuracy scores for cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 1 and target attributes being in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 1.6%. This indicates that the model was consistent. On average, the predictive model had an error rate of 88.8%.

Table 21 - Cross-validation results for k-nearest neighbour models trained with data stored using method 1 and target data in vector form

Fold #	Accuracy (%)
1	11.5
2	13
3	12.9
4	12.5
5	12.1
6	12.1
7	12.4
8	11.9
9	12.3
10	11.4
Average	12.2

4.4.4. Method 2 With Target Attributes in Vector Form

Table 22 shows the accuracy scores for cross-validation tests performed on a k-nearest neighbour model trained with data stored using method 2 and target attributes being in vector form. The range between the largest accuracy and the lowest accuracy obtained from the test is 2.8%. This indicates that the model was consistent. On average, the predictive model had an error rate of 90.3%.

Table 22 - Cross-validation results for k-nearest neighbour models trained with data stored using method 2 and target data in vector form

Fold #	Accuracy (%)
1	8.7
2	9.6
3	11.5
4	10.7
5	9.9
6	9.5
7	9
8	9.9
9	9.7
10	8.9
Average	9.7

4.5. Support Vector Machine Models

4.5.1. Method 1 With Target Attributes in String Form

Table 23 shows the accuracy scores for cross-validation tests performed on a support vector machine model trained with data stored using method 1 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 6.4%. This indicates that the model was consistent. On average, the predictive model had an error rate of 28.8%.

Table 23 - Cross-validation results for support vector machine models trained with data stored using method 1 and target data in string form

Fold #	Accuracy (%)
1	72.9
2	75.6
3	69.4
4	70.3
5	70.5
6	69.2
7	72.4
8	69.4
9	71.2
10	70.8
Average	71.2

Figure 25 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions at the beginning and end of the questionnaire also had relatively high error rates.

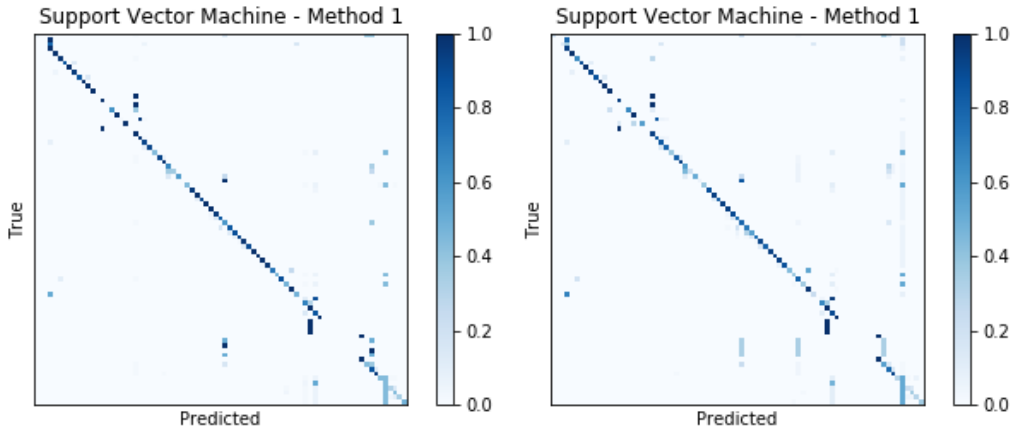

Figure 25 -Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a support vector machine model trained with data stored using method 1 and target attributes in string form

4.5.2. Method 2 With Target Attributes in String Form

Table 24 shows the accuracy scores for cross-validation tests performed on a support vector machine model trained with data stored using method 2 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 4.1%. This indicates that the model was consistent. On average, the predictive model had an error rate of 39.8%.

Table 24 - Cross-validation results for support vector machine models trained with data stored using method 2 and target data in string form

Fold #	Accuracy (%)
1	61.7
2	60.3
3	62.2
4	59.2
5	61.1
6	58.7
7	59.8
8	58.1
9	61.1
10	59.6
Average	60.2

Figure 26 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model failed to make accurate predictions. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used. Questions at the beginning and end of the questionnaire also had relatively high error rates.

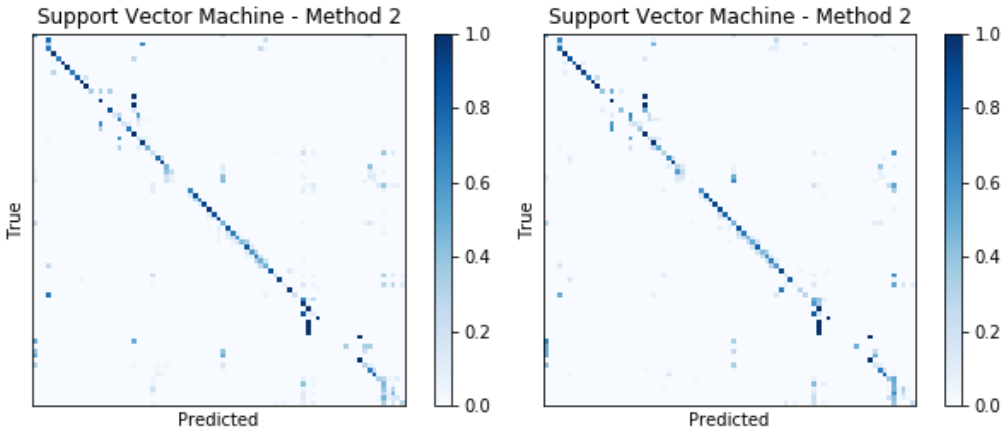

Figure 26 -Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a support vector machine model trained with data stored using method 2 and target attributes in string form

4.6. Naïve Bayes Models

4.6.1. Method 1 With Target Attributes in String Form

Table 25 shows the accuracy scores for cross-validation tests performed on a naïve Bayes model trained with data stored using method 1 and target attributes being in string form. The range between the largest and the lowest accuracy obtained from the test is 3.9%. This indicates that the model was consistent. On average, the predictive model had an error rate of 16.5%.

Table 25 - Cross-validation results for naive Bayes models trained with data stored using method 1 and target data in string form

Fold #	Accuracy (%)
1	85.4
2	84.8
3	81.8
4	82.9
5	84.1
6	82.1
7	85.4
8	81.5
9	83.8
10	82.9
Average	83.5

Figure 27 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model made false predictions. The results show that a model was also consistent in the type of false predictions it made. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used.

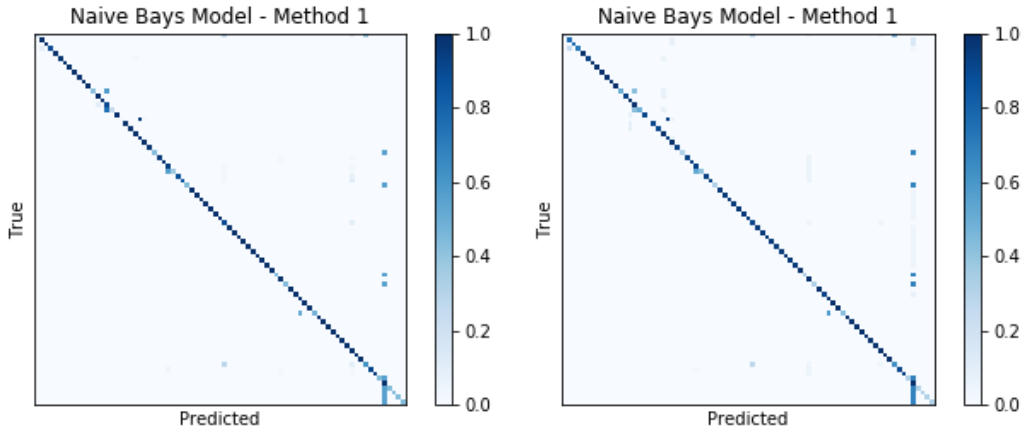

Figure 27 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a naïve Bayes model trained with data stored using method 1 and target attributes in string form

4.6.2. Method 2 With Target Attributes in String Form

Table 26 shows the accuracy scores for cross-validation tests performed on a naïve Bayes model trained with data stored using method 2 and target attributes being in string form. The range between the largest accuracy and the lowest accuracy obtained from the test is 2.6%. This indicates that the model was consistent. On average, the predictive model had an error rate of 21.1%.

Table 26 - Cross-validation results for naive Bayes models trained with data stored using method 2 and target data in string form

Fold #	Accuracy (%)
1	77.7
2	79
3	78.7
4	80
5	80.3
6	77.8
7	78
8	78.8
9	78.6
10	80.2
Average	78.9

Figure 28 shows the confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right). The two confusion matrices show consistencies in the points of the questionnaire where the model made false predictions. The results show that a model was also consistent in the type of false predictions it made. The results show that questions that followed branches in the question logic are the questions that cause the highest error rates for the predictive model used.

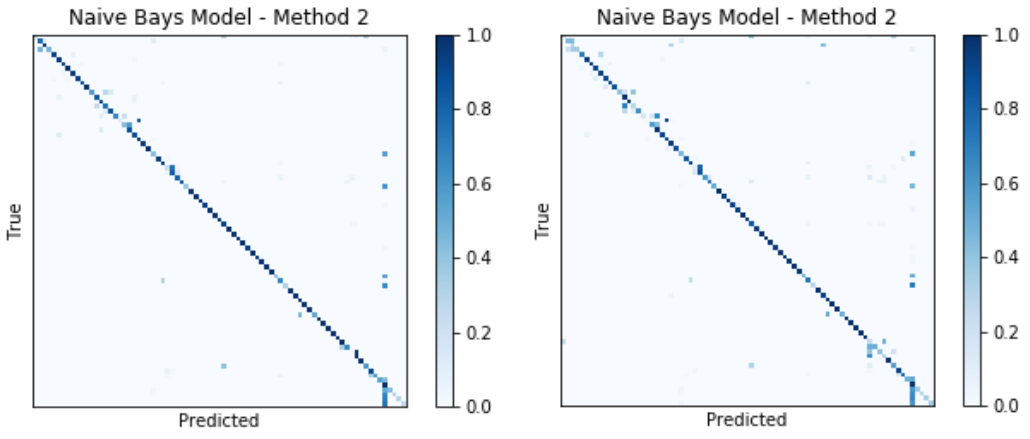

Figure 28 - Confusion matrices obtained for the fold that had the highest accuracy score (left) and the fold that had the lowest accuracy score (right) from cross-validation tests performed on a naïve Bayes model trained with data stored using method 2 and target attributes in string form

5. Discussion

For each of the cross-validation tests conducted during this project, the average accuracy score and the range between the highest and lowest accuracies were recorded for each modeling approach. This chapter analyses the trends and patterns that were observed from the results.

5.1. Comparison of Models' Accuracy Scores

5.1.1. Method of Storing Data

Methods 1 and 2 refer to the approach used to store users' data. In the first approach, user responses were stored at every point of the questionnaire and were applied to only 500 users. Method 2 involved storing user responses at randomly selected points of the questionnaire, but it was applied to all 2,397 users. Figure 29 shows the average accuracy scores obtained from every predictive model for each of the two methods when the target attributes were stored in string format.

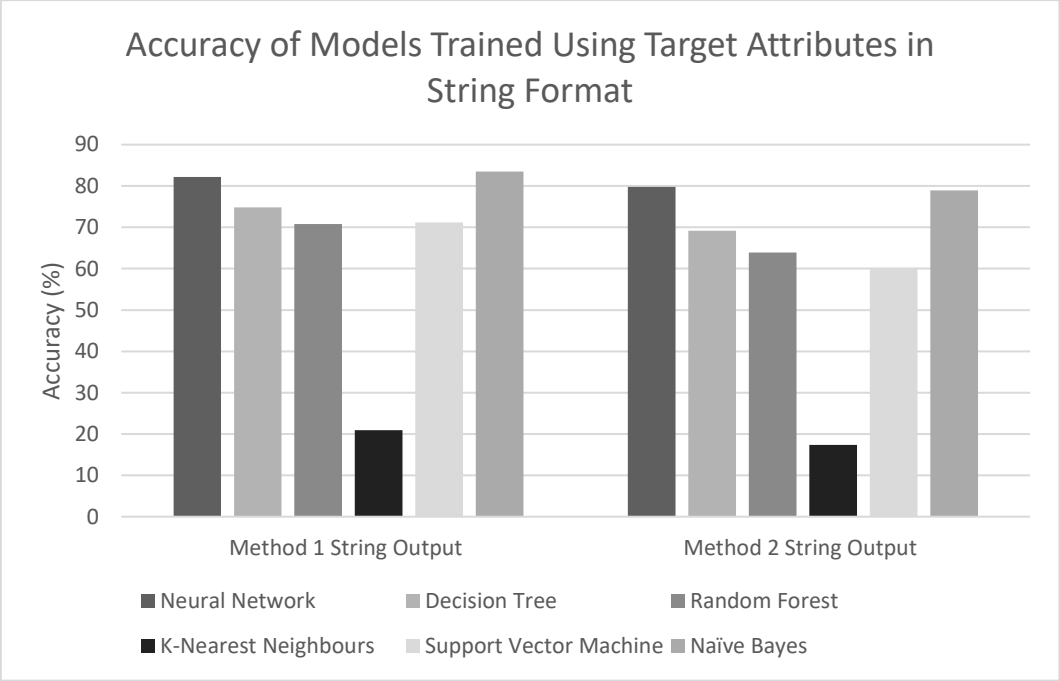

Figure 29 - Average accuracy scores obtained from every predictive model for each of the two methods, when the target attributes were stored in string format.

The accuracy was found to improve for all the predictive models when data was collected using the first method over the second method. The most probable cause of these results is that the first method contained fully itemised user questionnaires from each user in training sets. Even if questionnaires' data varied based on users' responses, a model would be exposed to some of these variations in the training phase. Using method 2 to store data meant that a model would be trained with data from more users, but there is a lower chance of the model being exposed to the full question logic. This meant that there is more chance of error at certain points of the questionnaire if the model was not trained with their occurrence.

This theory was tested by plotting the occurrence frequency of each question ID in the target attribute for both methods of storing data. The results are shown in Figure 30.

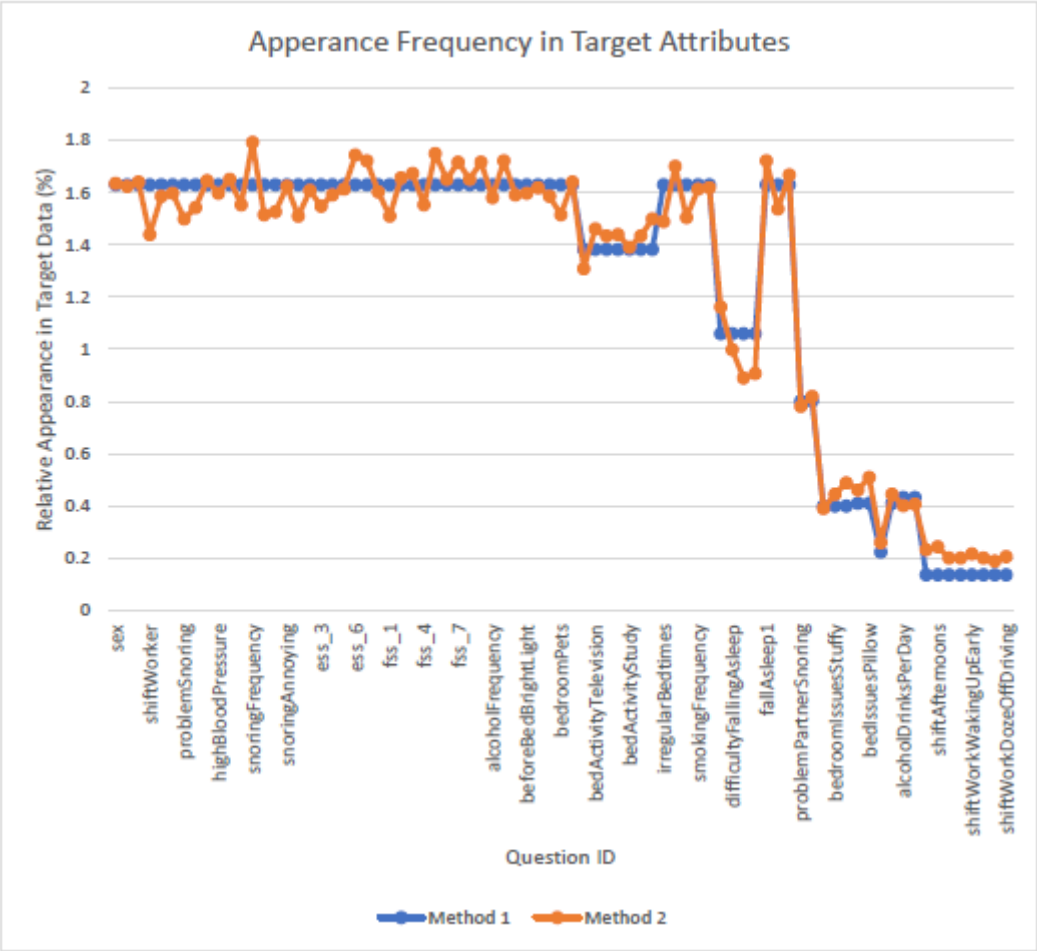

Figure 30 - Relative appearance of each question ID in target attributes

In Figure 30, the frequency of 1.63% indicates that a question was asked to every user in the dataset. This is clearly indicated for method 1. However, since the data stored was chosen randomly, using the second method meant that the relative frequency of appearance for each question was variable to inconsistency. This meant that some of the target attributes appeared in the data used to train the model more frequently than they would do in the actual questionnaire.

Splitting the data based on user IDs may have also contributed to the accuracy pattern. This is because, when method 1 is used, fully itemised questionnaires were contained for every user in the data set. This meant that the model was trained and tested with fully itemised questionnaires. While when method 2 is used, the dataset may contain more data from some users than others, therefore the model may have experienced a higher number of occurrences for certain parts of the questionnaire than others based on the cross-validation split.

Figure 31 shows the average accuracy scores obtained from every predictive model for each of the two methods when the target attributes were stored in vector format.

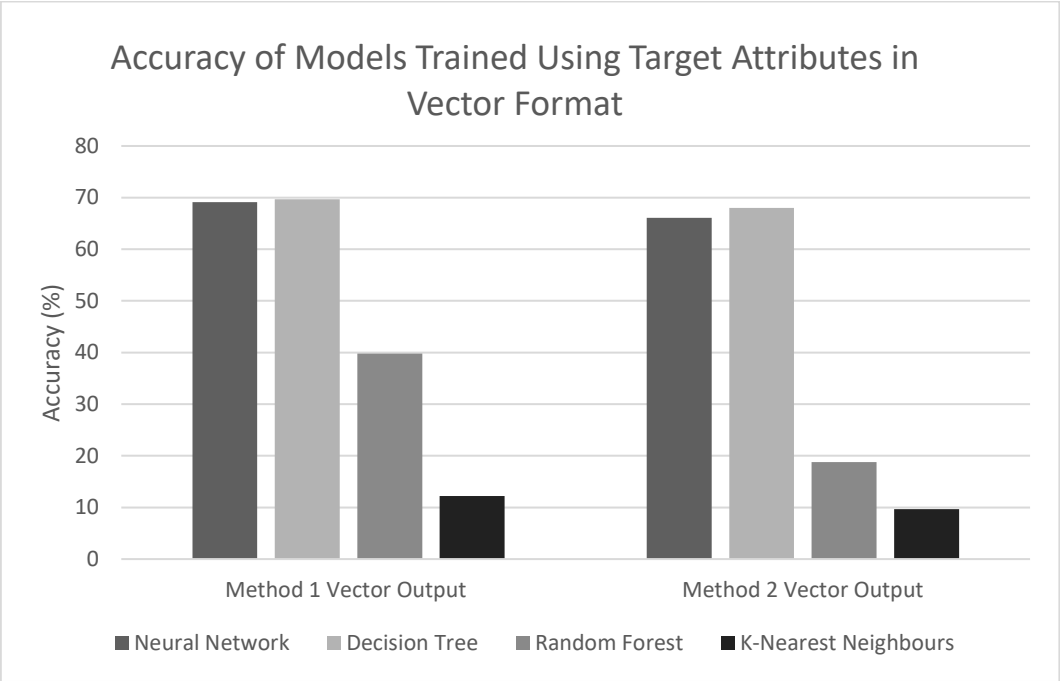

Figure 31- Average accuracy scores obtained from every predictive model for each of the two methods, when the target attributes were stored in vector format.

Figure 31 shows a similar trend to Figure 29, where all predictive models performed better when data was stored using method 1 over method 2. However, while the difference in accuracy for most predictive models is marginal, this was not the case for the random predictive model, where the use of method 1 to store data resulted in double the accuracy as method 2. This is likely caused by the fact that random forest is an ensemble predictive model that relies on the number of votes by a collective of decision trees to make a prediction. This shows that the use of method 2 to store data for vector formatted target attributes resulted in uncertainty that lead to inaccuracy of predictions. The use of a multilabel output could have contributed to this result.

5.1.2. Formatting of Target Attributes.

Two different ways were implemented in representing the questions that need to be asked, which are the target attributes in this experiment. One was by representing them using a vector of zeros and ones, and the other by using a string of the question ID. The method of representing the target attribute had an influence on the accuracy of predictive models. This is reflected in Figure 32 and Figure 33.

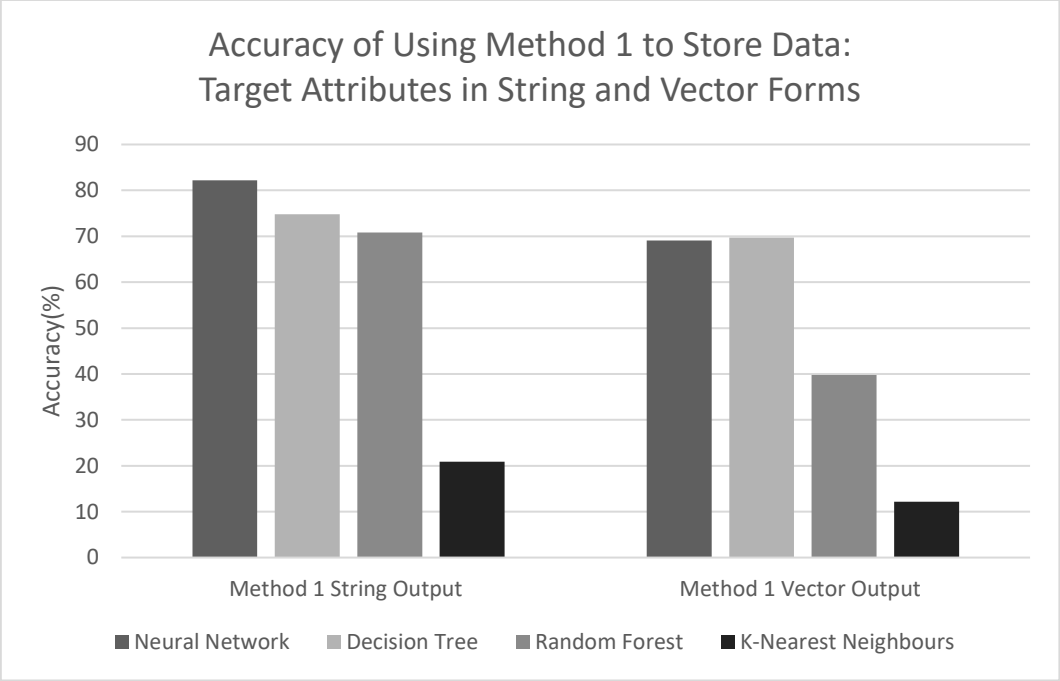

Figure 32 - Relationship between the format of target attributes and accuracy when method 1 is used to store data

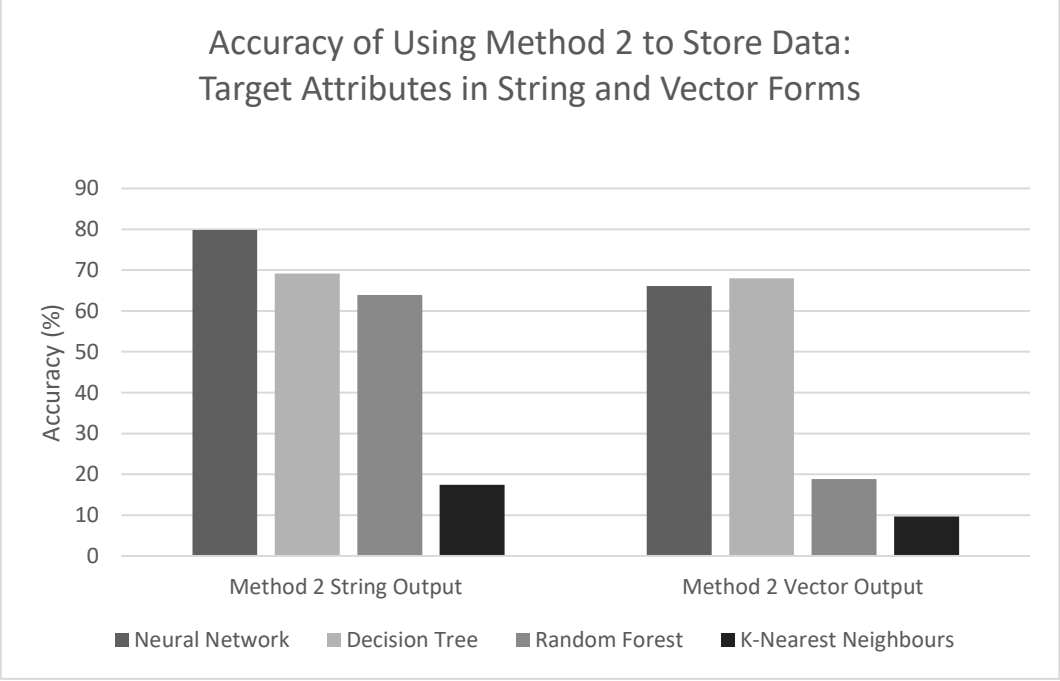

Figure 33 - Relationship between the format of target attributes and accuracy when method 2 is used to store data

As illustrated, when the target attributes were represented as strings the accuracy was higher than when they were represented as vectors. This is true regardless of the method used to store data or the predictive model used. However, while most predictive models suffered large reductions inaccuracy when target attributes were represented as vectors instead of strings, decision trees' accuracy was only marginally affected. This is likely due to the simplicity of the method used by decision tree models to make predictions. Since the method is binary, using multilabel values as target attributes does not affect the accuracy as it does for other models.

5.1.3. Comparison of Models

Table 27 shows the calculated average accuracy score for each of the predictive models used in this project.

Table 27 - Average accuracy of models based on the method of data storage and representation of target attributes.

	Neural Network (%)	**Decision Tree (%)**	**Random Forest (%)**	**K-Nearest Neighbours (%)**	**Support Vector Machine (%)**	**Naïve Bayes (%)**
Method 1 String Output	82.2	74.8	70.8	20.9	71.2	**83.5**
Method 2 String Output	**79.8**	69.2	63.9	17.4	60.2	78.9
Method 1 Vector Output	69.1	**69.7**	39.8	12.2		
Method 2 Vector Output	66.1	**68**	18.8	9.7		

Overall, the highest accuracy in predicting the questions was achieved when a naïve Bayes model was constructed using data stored using method 1 with target attributes in string format. However, the naïve Bayes model loses its status as the most accurate model when method 2 is used to store the data instead of method 1. This is likely due to the conditional independence in the design of naïve Bayes models. Which caused the model to thrive when trained with occurrences of fully itemised questionnaires.

Decision tree models had the highest accuracy scores when the target attributes were in vector form. This is likely due to the simplicity of the method used by decision trees to make predictions. That made it the most reliable model for multilabel output.

Neural network models had some of the highest accuracy scores amongst the predicting models used. This raises the questions on how its accuracy would be affected if its number of layers was changed for this application. Further research is needed.

5.2. Comparison of Consistency in Cross-Validation Tests

To test the consistency of each model the range from the cross-validation test accuracy score was computed. In this chapter, the consistency of the models is compared based on the method of storing data and the format of which target attributes were stored.

5.2.1. Method of Storing Data

Figure 34 shows the range for each model when target attributes were in string form. Models built with data stored using method 2 outperformed models built from data stored using method 1 in terms of consistency. This is likely caused by the fact that method 2 included data obtained from more users than method 1. That meant that when cross-validation was performed, method 2 was trained with data from more users and therefore more variations. However, method 1 showed higher accuracy as discussed previously.

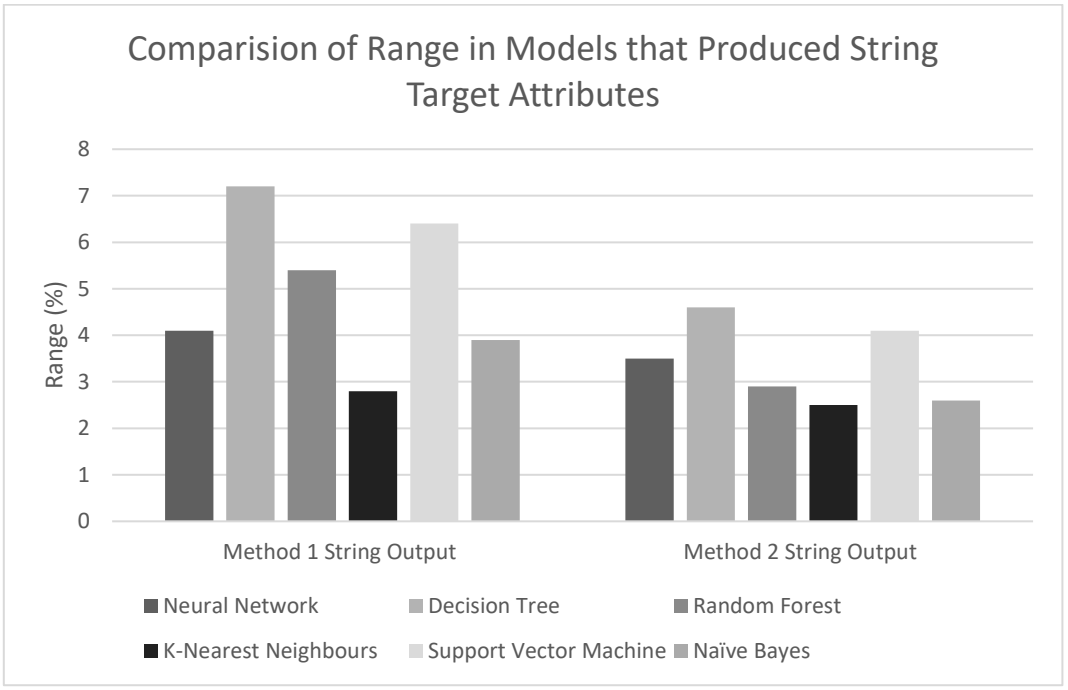

Figure 34-The effect of the method used to data on the consistency of predictive models when target attributes are in string format

Figure 35 shows the range for each model when target attributes are in vector form. For most models, the same results showed the same trend that appeared in Figure 34. One outlier was k-nearest neighbour, which had higher consistency for method 1 than method 2. This result was deemed insignificant due to the model's extreme inaccuracy.

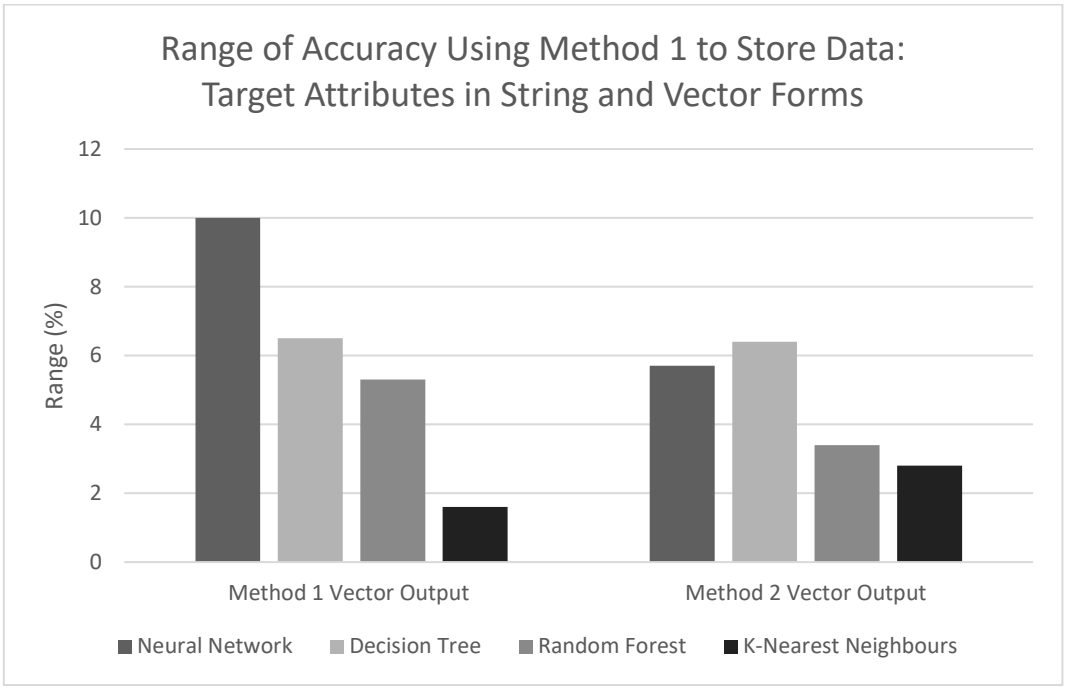

Figure 35 - The effect of the method used to data on the consistency of predictive models when target attributes are in vector format

5.2.2. Formatting of Target Attributes.

The method of representing the target attribute had an influence on the range of predictive models. As reflected in Figure 36 and Figure 37, using string representation caused higher consistency in the results.

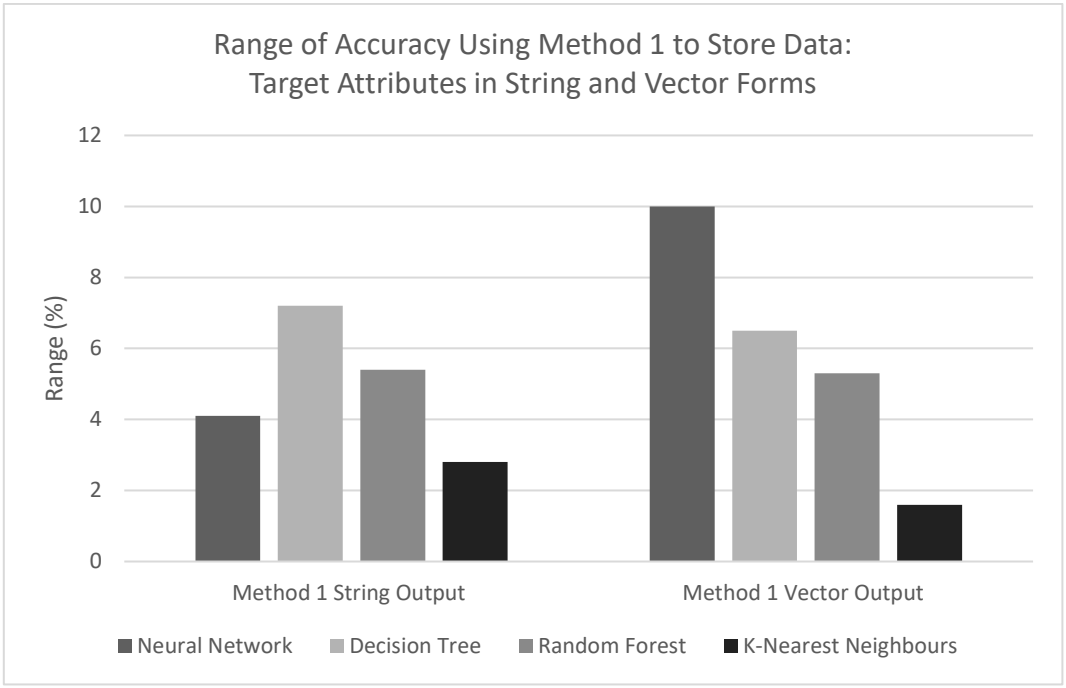

Figure 36- The effect of target attribute form on the consistency of models when method 1 is used to store the results

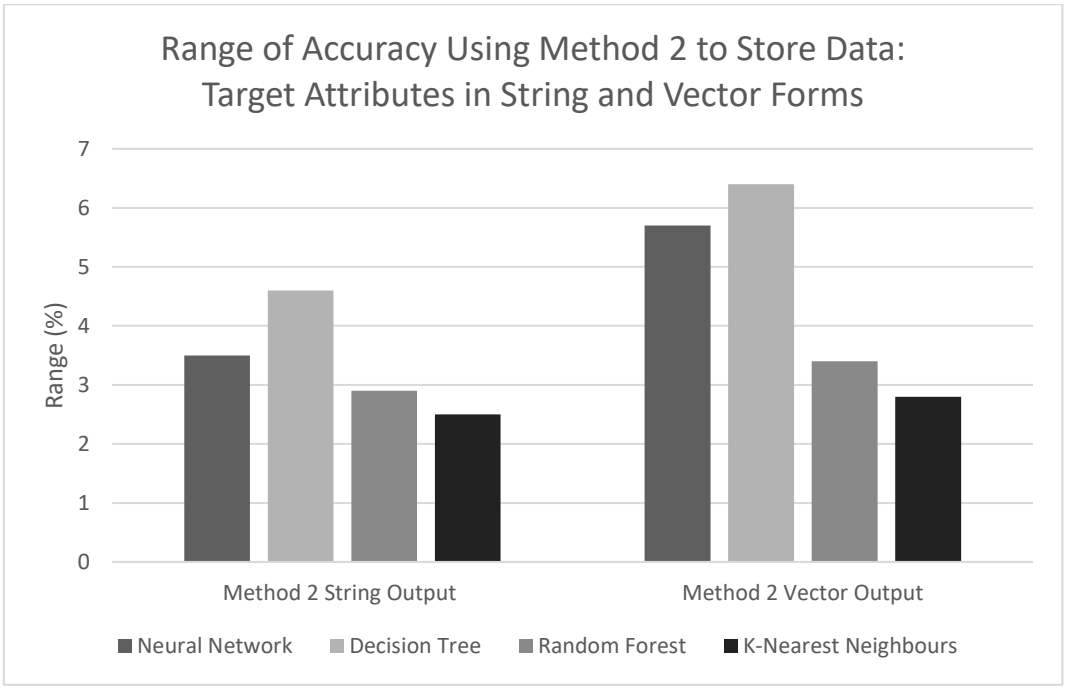

Figure 37 - The effect of target attribute form on the consistency of models when method 2 is used to store the results

5.2.3. Comparison of Models

Table 28 shows the calculated range for each of the predictive models used in this project.

Table 28 – Range of accuracy for models based on the method of data storage and representation of target attributes.

	Neural Network (%)	**Decision Tree (%)**	**Random Forest (%)**	**K-Nearest Neighbours (%)**	**Support Vector Machine (%)**	**Naïve Bayes (%)**
Method 1 String Output	4.1	7.2	5.4	**2.8**	6.4	3.9
Method 2 String Output	3.5	4.6	2.9	**2.5**	4.1	2.6
Method 1 Vector Output	10	6.5	5.3	**1.6**		
Method 2 Vector Output	5.7	6.4	3.4	**2.8**		

As illustrated, models that were most consistent were the k-nearest neighbour models. However, they were also the least accurate, making their consistency obsolete. When the output is in string form, naïve bays and neural network models had the highest consistencies and accuracies. This makes them the most desirable choice for this project.

5.3. Analysis of Confusion Matrices

Through analysis of the confusion matrices, it was found that the models constructed commonly share the type of errors they make. Especially for the accurate models, most of the errors occurred on questions that only were asked to users if they responded in a certain way to a previous question. An example of this is the question enquiring about users' alcohol frequency. The question has 5 answer choices, 'Never', 'Monthly of Less', 'Few per Month', 'Few Per Week' and, 'Lots per Week'.

By answering 'Few Per Week' or 'Lots per Week' a user is asked more questions about their alcohol drinking habits. Otherwise, the user is asked a different question, unrelated to alcohol consumption. Most models make errors on a question of this nature. For the alcohol frequency question, models fail to identify that users need to be asked further questions about their drinking habits when they respond with 'Few Per Week' or 'Lots per Week'. This is most likely caused by the fact that most users responded with one of the other three responses.

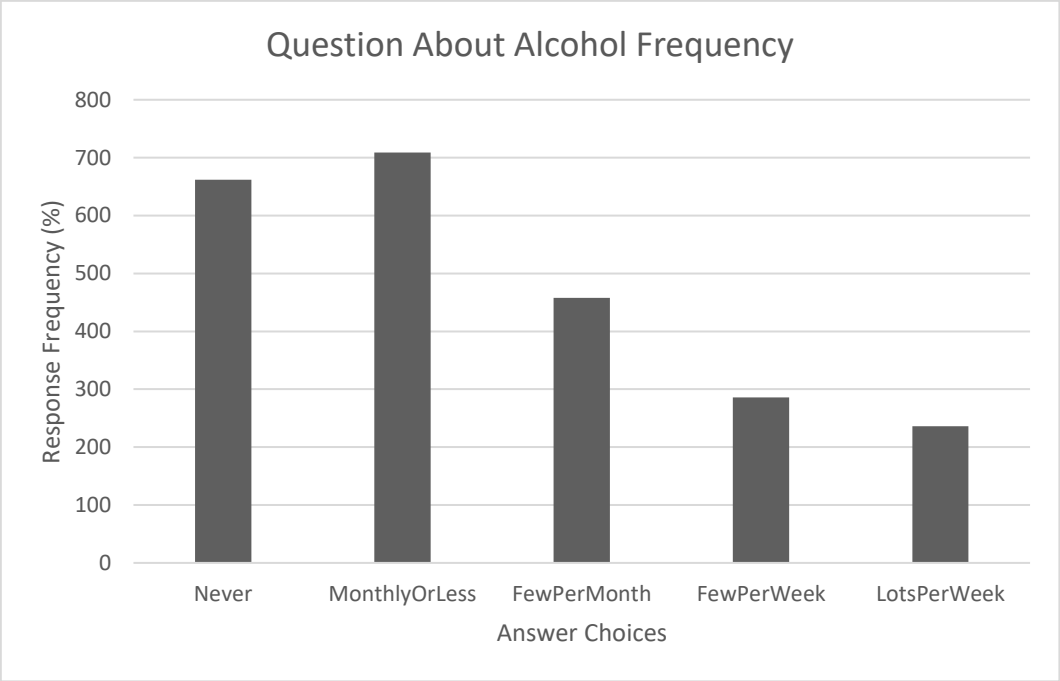

Figure 38- Users' responses to the question enquiring about alcohol frequency

6. Future Recommendations

Various steps could be taken to further understand the results of this project, apply them and then improve them. This chapter suggests some steps that can be taken to do that.

6.1. Inclusion of INT Questions in Modelling

While answers that were integers were not included in this experiment to avoid potential bias, that issue can be evaded if they were represented appropriately. One method that can be used to represent is by defining data ranges and representing them with vectors. An example of that is shown in Table 29. The ranges of data used could be determined depending on the distribution of the data, in a way that users' responses could be normalised easily.

Table 29 - The use of vectors to represent age ranges in datasets used for machine learning

User Response	Corresponding Vector
Ages 18-25	[0 0 0 1]
Ages 25-40	[0 0 1 0]
Ages 40-60	[0 1 0 0]
Ages above 60	[1 0 0 0]

6.2. Implementing Scikit-learn's 'predict_proba()' Function

During this project, Scikit-learn 'predict()' method was used to make predictions to get target data. However, the 'predict_proba()' function can be used to obtain a vector that contains a model's confidence score about each target attribute (Hackeling, 2017). Using those vectors, better methods can be utilised to make more accurate predictions. By obtaining the raw probability distribution of each target attribute, a set of rules can be defined to improve the accuracy of models.

6.3. Testing Models with High Accuracy Scores Further

Models such as neuron networks and naïve Bayes performed reasonably well for the identification of questions to ask. However, these models could be tested further to identify ways to improve them. For example, neural networks' models could be tested further by adding more layers and observing the effect on resulting models' accuracy scores then respond accordingly.

7. Conclusion

The aim of this project was to produce a machine learning model that can replicate the question logic of a sleep-health questionnaire. Various modeling techniques were employed in the process. Firstly, two different methods were used to store data to train the model. The first method relied on maximising the amount of data stored per user, while the second focused on maximising the number of users included in the model. Secondly, two different ways were employed in the representation of target data, one using vectors and the other using strings. Finally, various estimators were used to learn the question logic and make predictions accordingly. The results showed that the highest accuracy could be achieved with a Gaussian naïve Bayes model, that was trained with a dataset which maximises the amount of information contained per user, and target attributes represented as strings. A second model that produced remarkable results was a neural network model with a single hidden layer containing 173 nodes. Both models performed consistently.

Further research needs to be conducted to construct a model that can replicate the question logic. However, promising results were obtained in this project, which can later be developed and improved.

References:

Abernethy, J., Evgeniou, T., Toubia, O., Vert, J.-P., 2008. Eliciting Consumer Preferences Using Robust Adaptive Choice Questionnaires. IEEE Trans. Knowl. Data Eng. 20, 145–155. https://doi.org/10.1109/TKDE.2007.190632

Adams, R., Appleton, S., Taylor, A., Antic, N., 2017. Report to the Sleep Health Foundation 2016 Sleep Health Survey of Australian Adults 55.

Alpaydin, E., 2010. Introduction to machine learning, 2nd ed. ed, Adaptive computation and machine learning. MIT Press, Cambridge, Mass.

Blum, A.L., Langley, P., 1997. Selection of relevant features and examples in machine learning. Artif. Intell., Relevance 97, 245–271. https://doi.org/10.1016/S0004-3702(97)00063-5

Cheng, L., Adams, D.L., 1995. Yarn Strength Prediction Using Neural Networks: Part I: Fiber Properties and Yarn Strength Relationship. Text. Res. J. 65, 495–500. https://doi.org/10.1177/004051759506500901

Chervin, R.D., Hedger, K., Dillon, J.E., Pituch, K.J., 2000. Pediatric sleep questionnaire (PSQ): validity and reliability of scales for sleep-disordered breathing, snoring, sleepiness, and behavioral problems. Sleep Med. 1, 21–32. https://doi.org/10.1016/S1389-9457(99)00009-X

Chugh, R., 2013. Nested Refinement Types for JavaScript. UC San Diego.

DataCamp, n.d. Decision Tree Classification in Python [WWW Document]. URL https://www.datacamp.com/community/tutorials/decision-tree-classification-python (accessed 10.22.19).

Deng, Z., Zhu, X., Cheng, D., Zong, M., Zhang, S., 2016. Efficient kNN classification algorithm for big data. Neurocomputing, Learning for Medical Imaging 195, 143–148. https://doi.org/10.1016/j.neucom.2015.08.112

Dietterich, T., 1995. Overfitting and Undercomputing in Machine Learning. Comput. Surv. 27, 326–327.

El-Sayed, I.H., 2012. Comparison of four sleep questionnaires for screening obstructive sleep apnea. Egypt. J. Chest Dis. Tuberc. 61, 433–441. https://doi.org/10.1016/j.ejcdt.2012.07.003

File:Ann.png - Control Systems Technology Group [WWW Document], n.d. URL http://cstwiki.wtb.tue.nl/index.php?title=File:Ann.png (accessed 10.22.19).

Fontenla-Romero, Ó., Guijarro-Berdiñas, B., Martinez-Rego, D., Pérez-Sánchez, B., Peteiro-Barral, D., 2013. Online Machine Learning [WWW Document]. Effic. Scalability Methods Comput. Intellect. https://doi.org/10.4018/978-1-4666-3942-3.ch002

Garreta, R., Moncecchi, G., 2013. Learning scikit-learn: Machine Learning in Python. Packt Publishing Ltd.

Hackeling, G., 2017. Mastering Machine Learning with scikit-learn. Packt Publishing Ltd.

Hawkins, D.M., 2004. The Problem of Overfitting. J. Chem. Inf. Comput. Sci. 44, 1–12. https://doi.org/10.1021/ci0342472

Haykins, S., 2009. Neural networks and learning machines. Prentice Hall, New York.

Koehrsen, W., 2017. Random Forest Simple Explanation [WWW Document]. Medium. URL https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d (accessed 10.20.19).

Kortum, X., Grigull, L., Lechner, W., Klawonn, F., 2017. A Dynamic Adaptive Questionnaire for Improved Disease Diagnostics, in: Adams, N., Tucker, A., Weston, D. (Eds.), Advances in Intelligent Data Analysis XVI, Lecture Notes in Computer Science. Springer International Publishing, pp. 162–172.

Kotsiantis, S.B., 2007. Supervised Machine Learning: A Review of Classification Techniques, in: Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies. IOS Press, Amsterdam, The Netherlands, The Netherlands, pp. 3–24.

Krogh, A., 2008. What are artificial neural networks? Nat. Biotechnol. 26, 195–197. https://doi.org/10.1038/nbt1386

Kurhila, J., Miettinen, M., Niemivirta, M., Nokelainen, P., Silander, T., Tirri, H., 2001. Bayesian Modeling in an Adaptive On-Line Questionnaire for Education and Educational Research.

Langley, P., 2011. The changing science of machine learning. Mach. Learn. 82, 275–279. https://doi.org/10.1007/s10994-011-5242-y

McKinney, W., 2012. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc.
McKinney, W.G., 2010. Data Structures for Statistical Computing in Python.
Miettinen, M., Nokelainen, P., Kurhila, J., Silander, T., Tirri, H., 2005. EDUFORM – A Tool for Creating Adaptive Questionnaires. Int. J. E-Learn. 4, 365–373.
Motamedi, K.K., McClary, A.C., Amedee, R.G., 2009. Obstructive Sleep Apnea: A Growing Problem. Ochsner J. 9, 149–153.
Mwamikazi, E., Fournier Viger, P., Moghrabi, C., Baudouin, R., 2014. A Dynamic Questionnaire to Further Reduce Questions in Learning Style Assessment. Presented at the IFIP Advances in Information and Communication Technology, pp. 224–235. https://doi.org/10.1007/978-3-662-44654-6_22
Narkhede, S., 2019. Understanding Confusion Matrix [WWW Document]. Medium. URL https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (accessed 10.22.19).
Nishiyama, T., Mizuno, T., Kojima, M., Suzuki, S., Kitajima, T., Ando, K.B., Kuriyama, S., Nakayama, M., 2014. Criterion validity of the Pittsburgh Sleep Quality Index and Epworth Sleepiness Scale for the diagnosis of sleep disorders. Sleep Med. 15, 422–429. https://doi.org/10.1016/j.sleep.2013.12.015
Ohayon, M.M., Roberts, R.E., 2001. Comparability of Sleep Disorders Diagnoses Using DSM-IV and ICSD Classifications with Adolescents. Sleep 24, 920–925. https://doi.org/10.1093/sleep/24.8.920
Ortigosa, A., Paredes, P., Rodriguez, P., 2010. AH-questionnaire: An Adaptive Hierarchical Questionnaire for Learning Styles. Comput Educ 54, 999–1005. https://doi.org/10.1016/j.compedu.2009.10.003
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., 2011. Scikit-learn: Machine Learning in Python. Mach. Learn. PYTHON 6.
Ramachandran, S.K., Josephs, L.A., 2009. A Meta-analysis of Clinical Screening Tests for Obstructive Sleep Apnea: Anesthesiology 110, 928–939. https://doi.org/10.1097/ALN.0b013e31819c47b6
Senthilvel, E., 2011. Evaluation of Sleep Disorders in the Primary Care Setting: History Taking Compared to Questionnaires. J. Clin. Sleep Med. 7, 8.
Shalev-Shwartz, S., 2012. Online Learning and Online Convex Optimization. Found. Trends® Mach. Learn. 4, 107–194. https://doi.org/10.1561/2200000018
Singh, S., Gupta, P., n.d. Comparative Study Id3, Cart and C4.5 Decision Tree Algorithm: A Survey.
Talpur, A., 2017. Congestion Detection in Software Defined Networks using Machine Learning. https://doi.org/10.13140/RG.2.2.14985.85600
Zhang, H., 2004. The Optimality of Naive Bayes. Presented at the Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004.

Appendices

Appendix A: Code for Data Storage

```
from waleedsEdit import ProcessQuestions
from nextquestion import *
import unittest
import csv
import subprocess, os, re
import pandas as pd
import numpy as np
import json
import random

qnDef = pd.read_csv('engine/SleepCompanion/resultsManifest',
delimiter=':',skiprows=2, header=None)
options = dict()
dictofDictofVectors = dict()
pattern = re.compile('__[0-9]+')
for index, row in qnDef.iterrows():
    #     logging.info('index: %s row: %s' % (index, row))
    # then we want to create an enum for this one

    # import re, string
    try:
        dictName = pattern.sub('', row[0])
    except:
        print(0)

    try:
        row[3]
    except:
        print("logging")
    if row[3] in ['CHECKBOX', 'SINGLECHOICE']:
        # create a list of the options
        qnOptions = row[10].split(',')
        # then start building the dict
        optDict = dict()
        value = 0;
        for opt in qnOptions:
            vector=np.zeros(len(qnOptions))
            if opt == 'None':
                optDict[None] = vector
            vector[value]=1
            optDict[opt] = vector
            value = value + 1
        if None not in optDict.keys():
             # put in a default value for none
            optDict[None] = np.zeros(len(qnOptions))
        # evalStr = enumName + ' = OrderedEnum(\'' + enumName + '\',' +
str(optDict) + ')'
        options[dictName] = optDict
    else:
        options[dictName] = row[3]
questionIDs = list(options.keys())
vectorQuestions = dict()
for i in range(0,len(questionIDs)):
    vector=np.zeros(len(questionIDs))
    vector[i]=1
    vectorQuestions[questionIDs[i]] = vector

rawdata = pd.read_csv('ML/data/raw/completeUserData.csv')
questions =
"['sex','occupation','hasBedPartner','shiftWorker','shiftMornings','shiftAfternoons
','shiftEvenings','shiftExtendedHours','problemSleeping','problemSleepiness','probl
emSnoring','problemPartnerSnoring','beforeBedExercise','beforeBedWorkStudy','before
BedBrightLight','bedroomIssuesBright','bedroomIssuesStuffy','bedroomIssuesTemperatu
re','bedroomIssuesNoisy','bedroomIssuesNone','bedIssuesBlanket','bedIssuesPillow','
```

```
bedIssuesMatteress','bedIssuesPartner','badIssuesNone','bedroomPets','bedActivityTo
oLongRegularly','bedActivityTelevision','bedActivityRead','bedActivityEatDrink','be
dActivityStudy','bedActivityPhoneSocialMedia','bedActivityThink','bedActivityNone',
'irregularBedtimes','daytimeNaps','alcoholFrequency','alcoholDrinksPerDay','alcohol
ExcessiveFrequency','caffeinatedDrinksPerDay','smokingFrequency','difficultyDuratio
n','difficultyFallingAsleep','difficultyStayingAsleep','difficultyWakingTooEarly','
sleepPatternDissatisfaction','impactNoticeable','impactWorrying','impactInterfering
','ess_1','ess_2','ess_3','ess_4','ess_5','ess_6','ess_7','ess_8','fss_1','fss_2','
fss_3','fss_4','fss_5','fss_6','fss_7','fss_8','fss_9','sleepInitiation1','fallAsle
ep1','sleepInLater1','accidentRisk','shiftWorkWakingUpEarly','shiftWorkWellbeing','
shiftWorkDozeOff','shiftWorkDozeOffDriving','snoringBothersOthers','breathingStoppe
d','highBloodPressure','snoringOnBack','noseOftenBlocked','snoringFrequency','snori
ngDuration','snoringLoudness','snoringAnnoying','partnerSnoringBothersMe']"
data = {}
nextID = None
PQ = ProcessQuestions(questions=questions)
jsonData = []
singleRow = rawdata.iloc[1:5]
singleRow = singleRow.astype(str)
responses = singleRow.to_json(orient='records')

#ML= pd.DataFrame(columns=['userID','Features','Target'])
ML = pd.DataFrame()
features=pd.DataFrame()
target=pd.DataFrame()
if isinstance(responses, str):
    jsonData.extend(json.loads(responses.replace("'", '"')))
elif isinstance(responses, list):
    jsonData = responses
for i in range (0,len(jsonData)):
    nextID = None
    answers = None
    data = {}
    UIDs = list(jsonData[i].keys())
    responses = list(jsonData[i].values())
    while nextID != 'DONE!':
        vectorResponse = []
        questionVector = []
        numRows = len(ML)
        if nextID in UIDs:
            data[nextID] = responses[UIDs.index(nextID)]
            answers = '[' + (json.dumps(data).replace('"', "'")) + ']'
            unraveled = []
            unraveled.extend(json.loads(answers.replace("'", '"')))
            UIDsTwo = list(unraveled[0].keys())
            responsesTwo = list(unraveled[0].values())
            for w in range(0, len(questionIDs)):
                if questionIDs[w] in UIDsTwo:
                    location = UIDsTwo.index(questionIDs[w])
                    J =
(options.get(UIDsTwo[location])).get(responsesTwo[location])
                else:
                    J = (options.get(questionIDs[w])).get(None)
                vectorResponse = np.append(vectorResponse, J)
        elif nextID == None:
            for w in range(0, len(questionIDs)):
                J = (options.get(questionIDs[w])).get(None)
                vectorResponse = np.append(vectorResponse, J)
        nextID = PQ.calcNextQnID_back(None, answers)
        randomNum= random.randint(0,7)
        if (randomNum == 3):
            if nextID != 'DONE!':
                print(nextID)
                questionVector = vectorQuestions.get(nextID)
                ML.at[numRows, 'userID'] = responses[0]
                for j in range(0,len(vectorResponse)):
                    ML.at[numRows, j] = vectorResponse[j]
                    features.at[numRows, j] = vectorResponse[j]
```

```
                for j in range(0,len(questionVector)):
                    ML.at[numRows, questionIDs[j]] = questionVector[j]
                    target.at[numRows, questionIDs[j]] = questionVector[j]
                ML.at[numRows,'Target'] = nextID

ML.to_csv('engine/SleepCompanion/MLCompleteData', index=False)
features.to_csv('engine/SleepCompanion/featuresCompleteData', index=False)
target.to_csv('engine/SleepCompanion/targetCompleteData', index=False)
```

‘

Appendix B: Code for Machine Learning

```
import unittest
import csv
import subprocess, os, re
import pandas as pd
import numpy as np
from sklearn.model_selection import GroupShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.multioutput import MultiOutputClassifier
import matplotlib.pyplot as plt
from sklearn.utils.multiclass import unique_labels
from sklearn.neighbors import KNeighborsClassifier

ML = pd.read_csv('engine/SleepCompanion/ML', delimiter=',')
MLCompleteData = pd.read_csv('engine/SleepCompanion/MLCompleteData', de
limiter=',')
featuresCompleteData = pd.read_csv('engine/SleepCompanion/featuresCompl
eteData', delimiter=',')
features=pd.read_csv('engine/SleepCompanion/features', delimiter=',')
target=pd.read_csv('engine/SleepCompanion/target', delimiter=',')
targetCompleteData = pd.read_csv('engine/SleepCompanion/targetCompleteD
ata', delimiter=',')

print(features.shape)

print('Neural Network Model, Target data in string format, Method 1:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(features,ML['Target'] , groups=ML['userID'
]):
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=
(173,), random_state=1)
    X_train, X_test = features.iloc[train], features.iloc[test]
    y_train, y_test = ML['Target'].iloc[train], ML['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Neural Network Model - Method 1')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/NeuralNetworkStringMethod1_' + str(index) + '.
png')
```

```
    index = index + 1

print('Neural Network Model, Target data in string format, Method 2:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(featuresCompleteData,MLCompleteData['Targe
t'] , groups=MLCompleteData['userID']):
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=
(173,), random_state=1)
    X_train, X_test = featuresCompleteData.iloc[train], featuresComplet
eData.iloc[test]
    y_train, y_test = MLCompleteData['Target'].iloc[train], MLCompleteD
ata['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Neural Network Model - Method 2')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/NeuralNetworkStringMethod2_' + str(index) + '.
png')
    index = index + 1

print('Neural Network Model, Target data in vector format, Method 1:')
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(173
,), random_state=1)
scores = cross_val_score(clf, features , target, cv=10, groups=ML['user
ID'])
print(scores)

print('Neural Network Model, Target data in vector format, Method 2:')
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(173
,), random_state=1)
scores = cross_val_score(clf, featuresCompleteData , targetCompleteData
, cv=10, groups=MLCompleteData['userID'])
print(scores)

print('Decision Tree Model, Target data in string format, Method 1:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(features,ML['Target'] , groups=ML['userID'
]):
    clf = tree.DecisionTreeClassifier()
    X_train, X_test = features.iloc[train], features.iloc[test]
    y_train, y_test = ML['Target'].iloc[train], ML['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
```

```
    plt.title('Decision Tree Model - Method 1')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/DecisionTreeStringMethod1_' + str(index) + '.p
ng')
    index = index + 1

print('Decision Tree Model, Target data in string format, Method 2:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(featuresCompleteData,MLCompleteData['Targe
t'] , groups=MLCompleteData['userID']):
    clf = tree.DecisionTreeClassifier()
    X_train, X_test = featuresCompleteData.iloc[train], featuresComplet
eData.iloc[test]
    y_train, y_test = MLCompleteData['Target'].iloc[train], MLCompleteD
ata['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Decision Tree Model - Method 2')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/DecisionTreeStringMethod2_' + str(index) + '.p
ng')
    index = index + 1

print('Decision Tree Model, Target data in vector format, Method 1:')
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features , target, cv=10, groups=ML['user
ID'])
print(scores)

print('Decision Tree Model, Target data in vector format, Method 2:')
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, featuresCompleteData , targetCompleteData
, cv=10, groups=MLCompleteData['userID'])
print(scores)

print('Random Forest Model, Target data in string format, Method 1:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(features,ML['Target'] , groups=ML['userID'
]):
    clf = RandomForestClassifier(n_estimators=50, random_state=0)
    X_train, X_test = features.iloc[train], features.iloc[test]
    y_train, y_test = ML['Target'].iloc[train], ML['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
```

```
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Random Forest Model - Method 1')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/RandomForestStringMethod1_' + str(index) + '.p
ng')
    index = index + 1

print('Random Forest Model, Target data in string format, Method 2:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(featuresCompleteData,MLCompleteData['Targe
t'] , groups=MLCompleteData['userID']):
    clf = RandomForestClassifier(n_estimators=50,random_state=0)
    X_train, X_test = featuresCompleteData.iloc[train], featuresComplet
eData.iloc[test]
    y_train, y_test = MLCompleteData['Target'].iloc[train], MLCompleteD
ata['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Random Forest Model - Method 2')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/RandomForestStringMethod2_' + str(index) + '.p
ng')
    index = index + 1

print('Random Forest Model, Target data in vector format, Method 1:')
clf = RandomForestClassifier(n_estimators=50,random_state=0)
scores = cross_val_score(clf, features , target, cv=10, groups=ML['user
ID'])
print(scores)

print('Random Forest Model, Target data in vector format, Method 2:')
clf = RandomForestClassifier(n_estimators=50,random_state=0)
scores = cross_val_score(clf, featuresCompleteData , targetCompleteData
, cv=10, groups=MLCompleteData['userID'])
print(scores)

print('K-Nearest Neighbor Model, Target data in string format, Method 1
:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(features,ML['Target'] , groups=ML['userID'
]):
    clf = KNeighborsClassifier(n_neighbors=3)
```

```
    X_train, X_test = features.iloc[train], features.iloc[test]
    y_train, y_test = ML['Target'].iloc[train], ML['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('K-Nearest Neighbor Model - Method 1')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/KNNStringMethod1_' + str(index) + '.png')
    index = index + 1

print('K-Nearest Neighbor Model, Target data in string format, Method 2
:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(featuresCompleteData,MLCompleteData['Targe
t'] , groups=MLCompleteData['userID']):
    clf = KNeighborsClassifier(n_neighbors=3)
    X_train, X_test = featuresCompleteData.iloc[train], featuresComplet
eData.iloc[test]
    y_train, y_test = MLCompleteData['Target'].iloc[train], MLCompleteD
ata['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('K-Nearest Neighbor Model - Method 2')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/KNNStringMethod2_' + str(index) + '.png')
    index = index + 1

print('K-Nearest Neighbor Model, Target data in vector format, Method 1
:')
clf = KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(clf, features , target, cv=10, groups=ML['user
ID'])
print(scores)

print('K-Nearest Neighbor Model, Target data in vector format, Method 2
:')
clf = KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(clf, featuresCompleteData , targetCompleteData
, cv=10, groups=MLCompleteData['userID'])
print(scores)
```

```
print('Support Vector Machine Model, Target data in string format, Meth
od 1:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(features,ML['Target'] , groups=ML['userID'
]):
    clf = svm.SVC(gamma='scale')
    X_train, X_test = features.iloc[train], features.iloc[test]
    y_train, y_test = ML['Target'].iloc[train], ML['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Support Vector Machine - Method 1')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/SVMStringMethod1_' + str(index) + '.png')
    index = index + 1

print('Support Vector Machine Model, Target data in string format, Meth
od 2:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(featuresCompleteData,MLCompleteData['Targe
t'] , groups=MLCompleteData['userID']):
    clf = svm.SVC(gamma='scale')
    X_train, X_test = featuresCompleteData.iloc[train], featuresComplet
eData.iloc[test]
    y_train, y_test = MLCompleteData['Target'].iloc[train], MLCompleteD
ata['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Support Vector Machine - Method 2')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/SVMStringMethod2_' + str(index) + '.png')
    index = index + 1

print('Naive Bays Model, Target data in string format, Method 1:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(features,ML['Target'] , groups=ML['userID'
]):
    clf = GaussianNB()
    X_train, X_test = features.iloc[train], features.iloc[test]
    y_train, y_test = ML['Target'].iloc[train], ML['Target'].iloc[test]
```

```
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Naive Bays Model - Method 1')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/NBStringMethod1_' + str(index) + '.png')
    index = index + 1

Mprint('Naive Bays Model, Target data in string format, Method 2:')
index = 1
gss = GroupShuffleSplit(n_splits=10, test_size=0.1, random_state=0)
for train, test in gss.split(featuresCompleteData,MLCompleteData['Targe
t'] , groups=MLCompleteData['userID']):
    clf = GaussianNB()
    X_train, X_test = featuresCompleteData.iloc[train], featuresComplet
eData.iloc[test]
    y_train, y_test = MLCompleteData['Target'].iloc[train], MLCompleteD
ata['Target'].iloc[test]
    y_pred=clf.fit(X_train, y_train).predict(X_test)
    print(accuracy_score(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    im=plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    if index == 1:
        plt.colorbar(im)
    plt.title('Support Vector Machine - Method 2')
    plt.ylabel('True')
    plt.xlabel('Predicted')
    plt.yticks([])
    plt.xticks([])
    plt.savefig('results/SVMStringMethod2_' + str(index) + '.png')
    index = index + 1
```