MSc Thesis: Developing a Recommendation Algorithm for Patients using the Healthentia Platform

Alex Bujorianu
Business Information Technology: Data Science
University of Twente
Student no. s2451980
a.bujorianu@student.utwente.nl

Prof. Dr Peter Lucas, MD
First Supervisor
EEMCS, University of Twente
p.j.f.lucas@utwente.nl

Saskia Kelders, PhD
Second Supervisor
BMS, University of Twente
s.m.kelders@utwente.nl

Harm op den Akker, PhD
Industry Supervisor
Innovation Sprint
hopdenakker@innovationsprint.eu

September 9, 2022

UNIVERSITY
OF TWENTE.

Executive Summary

The purpose of this work is to develop a high-level recommendation algorithm that provides medical advice to users of the Healthentia Virtual Coaching platform. The algorithm is tasked with choosing the most relevant recommendations from a set of 11 possible recommendations. For example, a patient that suffers from obesity may be told to eat less, consume more fibre and exercise more. The algorithm must not provide generic or irrelevant advice, but should take into account data about the patient and choose only the most relevant recommendations. Formally, this is treated as a multi-label classification problem.

Training data is generated using a proprietary simulation part-developed by the author, and manually labelled. A variety of machine learning algorithms are trained and their performance evaluated on the training set using cross-validation. In addition, a "knowledge algorithm" is created by formalising domain knowledge rules. The results indicate that the knowledge algorithm performs the best.

Additionally, a number of practical concerns are also discussed; the knowledge algorithm is the best option in terms of development effort (also called time to Minimum Viable Product), modifiability, and explainability. It also has a fast prediction time, making it very scalable. The problem of data collection and imputation is also considered: the advantage of the knowledge algorithm is that it can handle missing data relatively easily, without requiring the use of multiple models for each missing feature.

In terms of its contribution to the literature, this work emphasises the fact that rules-based knowledge algorithms can perform better than learning-based approaches, provided that the problem is well-understood by domain experts and formalisable. Given the disadvantages of supervised learning-based approaches – sensitivity to unbalanced data, labelling requirements, overfitting, training/optimisation time, and sensitivity to missing data – knowledge-based algorithms provide a compelling alternative. This is particularly true of the medical field, where labelling often needs to be done by experts, and data is frequently incomplete.

Keywords: eHealth, Supervised Machine Learning, Domain Knowledge, Rules-based Algorithms, Simulation

Contents

1 Introduction **7**

2 Related Research **11**

3 Methods **13**
3.1 Generation of Synthetic Data 13
3.2 Visualisations 15
3.3 Sleep 16
3.4 Diet 17
3.5 Body Fat Percentage vs BMI 18
3.6 Exercise 19
3.7 Scope of Recommendations 19
3.8 List of Recommendations 21
3.9 Labelling of the data 22
3.10 Ranking of recommendations 23
3.11 Visualisation of labels 24
3.12 Model Selection 24
3.13 K-fold Cross Validation 26
3.14 Scaling 26
3.15 Hyperparameter Optimisation 27

4 Outcome Measures **32**
4.1 Inter-annotator metrics 33
4.2 Strict accuracy 33
4.3 Hamming Loss 33
4.4 Partial accuracy 34
4.5 Label accuracy 34
4.6 Length Mismatch & Length Ratio 34
4.7 Confusion Matrix 35

5 Results **35**
5.1 Interannotator reliability 35
5.2 Partial accuracy 35
5.3 Strict Accuracy 36
5.4 By Label 36
5.5 Length Mismatch and Length Ratio 39
5.6 Confusion Matrices 40
5.7 Applicability to different human coders 41
5.8 Benchmarks 41

6 Discussion **42**
6.1 Which algorithm has the best performance? 42
6.2 Introducing new recommendations 43
6.3 Explainability 43
6.4 Is domain knowledge superior to ML? 44
6.5 Further testing 45
6.6 Limitations 46
6.7 Imputation 46
6.8 Conclusion 48

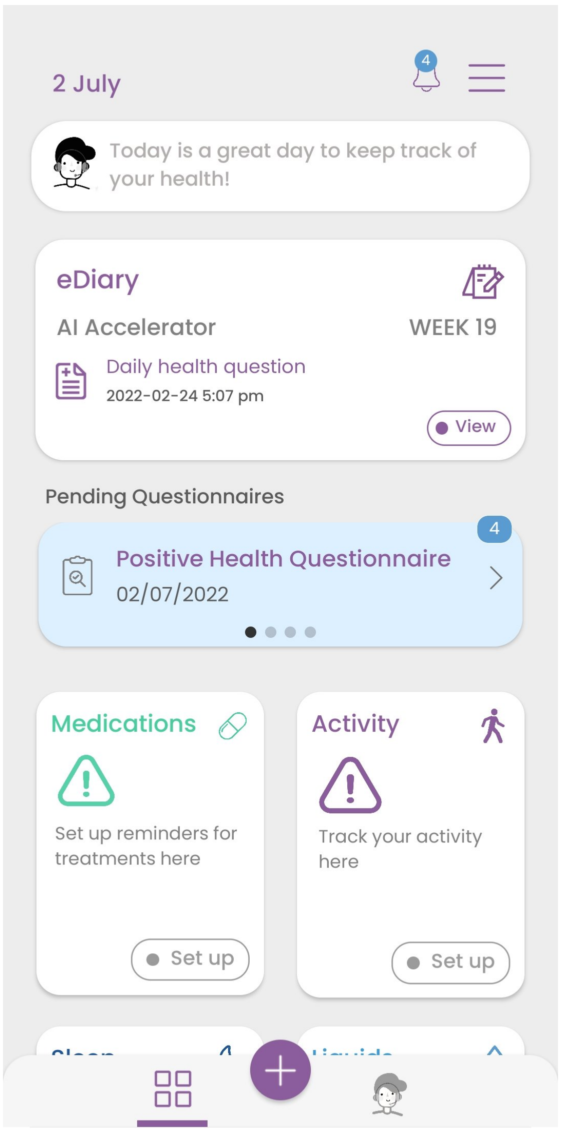
List of Acronyms

- BMI : Body Mass Index. The patient's weight divided by the square of their height.
- CHD : Coronary Heart Disease. An obstruction of the arteries by cholesterol plaques.
- COPD : Chronic Obstructive Pulmonary Disease. A chronic inflammatory disease of the lungs that causes obstruction.
- DEXA : Dual-Energy X-ray absorbptiometry. A type of X-ray that is used to measure a patient's body fat.
- DHIs : Digital Health Interventions.
- GI : Glycemic Index. A measure of how rapidly carbohydrates are converted into blood glucose. Values range from 0 to 100.
- GIGO : Garbage In Garbage Out
- GL : Glycemic Load. The amount of carbohydrates multiplied by the glycemic index determines how much blood sugar levels will rise.
- HDL : High Density Lipoprotein. A type of cholesterol with a protective effect on the heart.
- JSON : Javascript Object Notation. A data storage format.
- KNN : K nearest neighbours. A type of distance-based classification algorithm.
- LBFGS : Limited-memory BFGS. An alternative to stochastic gradient descent optimisation.
- LDL : Low Density Lipoprotein. The "bad" kind of cholesterol that causes heart disease.
- LLOC : Logical Lines of Code. The number of programming statements in a file of code (as opposed to comments).
- MAP: Maximum a Posteriori. A statistical technique that involves the calculation of a conditional probability given a model weighted by a prior belief about the model. It is used by multi-label K-nearest neighbours.

- MCAR : Missing Completely at Random. The missing values are independent of other variables.
- ML : Machine Learning. An automated way of mapping inputs to correct target outputs.
- MLKNN : Multi-Label K nearest neighbours.
- MRI : Magnetic Resonance Imaging.
- MVP : Minimum Viable Product.
- NHS : National Health Service (of the UK)
- ReLU : Rectified Linear Unit. A type of activation function used in deep neural networks. ReLU(x) = x if x is greater than 0 else 0.
- SARIMA : Seasonal Autoregressive Moving Average. A time-series model used to predict seasonal data.
- SGD : Stochastic Gradient Descent. A numerical optimisation method that is used to find the best weights of a neural network.
- SQL : Structured Query Language. A language for querying databases.
- SVM : Support Vector Machine. A type of classification algorithm.
- TAM: Technology Acceptance Model. A psychological model that represents the person's intention to use a technology. It is typically used by researchers in the IT domain.
- UTAUT: Unified Theory of Acceptance and Use of Technology: A competitor to TAM that focuses on different psychological concepts.

1 Introduction

The Healthentia mobile application, developed by Innovation Sprint, is a class-I medical device that is currently used by doctors to collect data for clinical trials (figure 1). The purpose of this work is to investigate how the planned Virtual Coaching features could be delivered by the Healthentia app. Virtual Coaching is a new field that combines technology, medicine and applied psychology in order to deliver personalised lifestyle advice; it usually encompasses the areas of diet, exercise and sleep; may incorporate mental health; and may also include more specialised advice for patients with chronic conditions like diabetes, heart disease and Chronic Obstructive Pulmonary Disease (COPD). The idea behind Virtual Coaching is that the program can interact with the user in a similar way to a personal trainer or a doctor. The app not only gives advice, but it may also hold "conversations" with the user, or provide encouragement. The advice comes in two forms: day-to-day advice, and general advice that is meant to be followed for a longer period of time. So, for example, the patient may receive the advice to consume more fibre; the day-to-day algorithm may suggest eating oats for breakfast. One can also think of these algorithms as carrying out "diagnosis" and "treatment plan".


2 July

Today is a great day to keep track of your health!

eDiary

AI Accelerator WEEK 19

Daily health question

2022-02-24 5:07 pm

Pending Questionnaires

Positive Health Questionnaire

02/07/2022

Medications

Set up reminders for treatments here

Activity

Track your activity here

Figure 1: Screenshot of the Healthentia mobile application on Android.

For the purpose of my thesis, I am concerned only with developing the long-term diagnostic part of the program, which I term the recommendation algorithm.

What is the rationale for building a digital health intervention system, like virtual coaching? The burden of chronic disease has increased dramatically throughout the world since 1970. The majority of chronic diseases are caused, at least in part, by lifestyle factors, and are therefore, to some extent, preventable (see [1] for a discussion). Examples include diabetes; atherosclerosis; a weak heart muscle; COPD; skin cancer; obesity-related morbidities (e.g. lower back pain) and to a lesser extent, cancers of all types. Obesity and diabetes, in particular, have seen a dramatic rise in prevalence over the past decades: obesity has gone from 13% in 1970 to 36% in 2010 [2]. It has since increased to 41%. [3] Diabetes has been following a similar trend [4].

The key to tackling these chronic diseases is behaviour: eating less, eating better, and moving more. Unfortunately, it is very hard to get people to change their behaviour. This is because there are powerful physiological mechanisms at play that make it difficult to lose weight and cut sugar consumption. For example, Bray presents a number of physiological arguments for why sugar should be considered addictive [5]. Jastreboff et al. have shown that teenagers who are obese have a different brain response to sugar when compared to teenagers who are of normal weight [6]. Furthermore, Drenowski [7] has shown that naxolone can inhibit binge-eating. Although it may be tempting to think that pharmacological interventions can substitute for behavioural change (the colloquial "magic pill") the unfortunate reality is that appetite suppressants have nasty side-effects on cardiovascular health [8, 9]. Moreover, they are not effective enough on their own to prompt a substantial improvement in weight; at best, they can be coupled with behavioural interventions to provide encouraging weight loss in the short-term.

Digital health intervention systems (DHIs) are promising because they may be more effective than traditional approaches at persuading people to make changes, owing to two characteristics. Firstly, DHIs have the characteristic of being always-on, whereas a patient might only see a doctor once a month. Secondly, DHIs can adapt to user input and behaviours, making them a type of personalised healthcare. While in principle doctors can do the same thing with patients they see often, in reality, the large number of patients makes it difficult for them to do this in practice. Research has shown that personalised healthcare can sometimes be more effective than one-size-fits-all approaches. [10]

Even if automated recommendation algorithms have only a modest impact in reducing the incidence of new chronic diseases, as well as managing existing conditions, the economic savings are huge. The NHS spends [11] £10 billion pounds a year on treating diabetes-related morbidity – that's nearly 10% of its entire budget. If a recommendation algorithm can prevent 10% of diabetes cases then, over time, it could save the NHS £1 billion a year. When considered on a global scale, electronic health intervention systems like virtual coaching are a multi-billion euro industry: McKinsey estimates that the sector is worth $350 billion. [12]

It is important to note that the numbers above are only rough estimates, and should be treated with a tablespoon of salt. So far, there is very limited data quantifying the cost savings of eHealth interventions, and this is partly because of methodological challenges [13]. The eHealth sector is very diverse in terms of technologies used, the diseases being tackled, and the outcome measures; it is also changing rapidly. In other words: at present, nobody really knows how effective virtual coaching is at saving money. I use the *potential* financial savings as a justification for this proof-of-concept work, but I do not attempt to quantify how effective this system would be and how that would translate into financial savings. That is an area for future research.

One of the main technical questions addressed by this work is whether the approach to giving recommendations should be based on Machine Learning, or whether a so-called "knowledge-based algorithm" is the better choice. This is a design decision with important consequences for how well the algorithm will work, how easy it is to understand, and how much time is required to develop it.

What is a knowledge algorithm, and how does it differ from the machine learning approaches that are common today? The difference lies in how a knowledge base is acquired. Using machine learning, the knowledge is "learned" from the data. For example: if we have a group of patients that are labelled healthy, and another group that are labelled unhealthy, and only one feature, BMI, the ML algorithm can infer that patients with a BMI greater than 30 are unhealthy, and those with a BMI less than 30 are healthy. Very simple algorithms like linear perceptron and nearest-neighbours are good at making this kind of classification.

With a knowledge-based algorithm, however, we already know that healthy patients have a BMI less than 30 because doctors have documented this fact in the literature. This classification can therefore be performed with a simple if-else statement.

The advantage of machine learning is two-fold. Firstly, it does not require the programmer to have deep domain knowledge about the problem area; the programmer's only responsibility is to choose the right algorithm that will arrive at a good solution. Knowledge-based algorithms, on the contrary, rely on a deep understanding of the underlying problem [14]. Thus, ML is advantageous when the knowledge of the problem area does not exist. Secondly, even when the knowledge is available, it is not necessarily easy to formalise in a classical rules-based algorithm. For example, interaction effects are not easy to model, and they may not be (fully) known even by domain experts. Machine Learning approaches, on the other hand, are good at handling this.

However, knowledge-based algorithms have a number of practical advantages. They do not require labelled data (so-called training sets) – which is a major plus, as labelling can take a long time. Knowledge-based algorithms are less sensitive to unbalanced data, whereas ML algorithms tend to perform well on more frequent labels and poorly on less frequently-encountered labels. Nevertheless, it was not possible to know *a priori* whether a knowledge algorithm would perform better than a ML algorithm.

Regardless of what approach is chosen, whether it be machine learning or a knowledge-based algorithm, the architecture of the recommender system will basically be the same. What does this mean? It means that the choice of algorithm for the classification task does not affect the choice of algorithm for the coaching task, since the interface stays the same. It is useful to refer to figure 5.

Goals The recommendation algorithm should fulfill the following criteria in order to be considered a good solution:

1. Its performance must match, or at least approach, that of a human. For an indication of how well a human performs, see the section titled Interannotator reliability.

2. It must be explainable to humans, in order for doctors to entrust the algorithm with their patients. Why did the algorithm make the decision that it did?

3. It must be cost-effective to develop in a reasonable amount of time, e.g. a few months. Ehealth is a fast-moving field and the firm's budget is limited.

4. It must have low latency (less than 0.1ms per patient) and be able to handle a reasonably large number of users, anywhere from e.g. 10K to 100K users. The algorithm is going to run server-side.

2 Related Research

Although a huge amount of work has been carried out to develop recommendation algorithms, primarily by media giants like Netflix, Google and Amazon, the actual implementation details of most of these algorithms are trade secrets. This is also the case in the eHealth domain. The problem of multi-label classification is a generally well-studied one, which is why I have chosen to use it; the difference lies in the context. Most multi-label classification algorithms have been used for text or movie categorisation (see e.g. [15], and [16]). They have not typically been applied to medical uses. This is not necessarily an issue; there are a wide variety of approaches used in the eHealth domain, and because the field is so new, many of them are untested.

Stark et al. [17] conducted a thorough literature review on the state-of-the-art in medical recommender systems. They included papers written after the year 2000 which were published in ACM Digital Library, IEEExplore, ScienceDirect, Elsevier, John Wiley or Google Scholar. This resulted in a total of 52 papers; after some more criteria were applied, the authors narrowed it down to 13 papers. What is most interesting about this literature review is the wide variety of methods that were used in each paper, which can be categorised into three main approaches. The first, rules- and ontology-based algorithms, is similar to the knowledge algorithm that I develop in this thesis. The second approach is to use supervised machine learning algorithms, like SVM and neural networks, which are also used in this paper. The third approach focuses on unsupervised learning methods, though this is primarily for feature engineering, i.e. the output of an unsupervised algorithm is typically used as the input for a supervised learning algorithm. This study does not directly provide input on the technical question being addressed, rather, it illustrates the fact that many approaches are possible, and the choice of algorithm is very situation-dependent.

The outcomes and disease types were extremely varied, but diabetes and drug interactions were well-represented. In terms of software engineering, integration challenges (that is, interoperability between different databases and systems used in hospitals, especially legacy software) and scalability were identified as key problems. Indeed, the motivation for developing these algorithms in the first place is usually to treat large numbers of patients for whom doctors have to prescribe medications, as this can save valuable time. The IRS-T2D study [18] is a prime example of this. Another motivation for developing these systems is to account for the large number of medications that are available on the market; some may are more appropriate for patients with certain conditions than others, and there are often complex interaction effects between other medications, which are not sufficiently well-understood by doctors.

The DiaTrack study [19] is particularly interesting, as it uses a more novel approach: it searches for diabetes patients that are most similar to a patient, in order to optimise the recommendations (drug selection and treatment regimen). It does this not through conventional means like clustering algorithms, but using linked data, which is mainly useful for simplifying database design. Another study, T-Recs [20], trained a decision tree classifier on tweets (from the Twitter social media platform) to provide appropriate advice, taking into account patient comorbidities, symptoms, duration, and sentiment. What's unusual about this study is both the source of data and the inclusion of sentiment analysis/NLP as a higher-dimensional input.

Irrespective of the approach chosen, the basic challenges are the same: a large number of patients requiring scalability; interoperability with different systems; and the need to take into account complex interactions between the recommendations and patient characteristics. The first and third challenges are addressed in this work as well. What the study also shows is that the best solution is highly dependent on the problem, and there is no one-size-fits-all approach.

3 Methods

This section covers a wide range of methods that were used for several tasks: generating realistic data using a simulation; labelling said data; selecting the right models; and optimising the models. Each subsection starts with a high level overview and progressively becomes more detailed.

3.1 Generation of Synthetic Data

The Healthentia app is sold as a software service (SaaS) for doctors to collect patient data with. As such, Innovation Sprint does not own the rights to the data collected by their partners; they are considered data processors. Therefore, a synthetic dataset had to be generated. In order to do this, a proprietary in-house simulation was modified. The purpose of this section is to describe what data is generated, how, and why. The simulated data had to fulfill three criteria:

Available. It must be possible for the data to be collected by the Healthentia mobile app in some fashion, either through wearables or questionnaires. Sleep duration and sleep quality can be determined using a Fitbit or similar device. Some exercise data (e.g. number of steps, duration and intensity of exercise) can also be obtained using these devices. Data about diet has to come from self-reporting.

Realistic. The data has to be a good match, statistically, with real data, and it must incorporate realistic relationships between variables, such as diabetes and obesity.

Clear mapping. The data must make sense given our recommendations, and individual input variables should map onto recommendations in a clear way. (This criterion, if fulfilled, implies that a knowledge algorithm would perform well.) Variables which are interesting to humans can be simulated, but such "nuisance" variables should be removed before being passed to the recommender algorithm – which is exactly what I do (see the `get_data` function in the code). The same applies to real data which collects variables of interest to researchers or patients. The reason it is important to do this is because adding more variables to a ML algorithm increases the number of parameters, and thus makes it harder to train.

The simulation creates patients and simulates their life for three months. Each patient is independent from another (they do not interact). The simulation is imperfect because it does not model long-term relations between variables. For example: the patient's intake of saturated fat does not determine their lifetime risk of developing coronary heart disease (CHD). CHD occurs because of a process called atherosclerosis, whereby plaques begin to accumulate in the major arteries, caused by a combination of many Low-Density Lipoprotein (LDL) cholesterol particles and the inflammatory secretions of senescent cells. However, since we only simulated a relatively short portion of the patient's lives, this was not a priority to implement – atherosclerosis is a process that occurs over many years. Likewise, it should be noted that a patient's weight does not vary with their calorie intake – patients with a high BMI are assumed to eat a large number of calories.

The patients may also start with chronic diseases, which are determined by random draws at the beginning of the simulation, and considered incurable for the remainder of the simulation. For example, some patients will be classed as type-2 diabetics. This is based on their sex and BMI:

	Underweight	Healthy Weight	Overweight	Obese
Female	1.9%	1.9%	4.3%	10.7%
Male	0%	3.3%	6%	14.6%

Prevalence of diabetes by weight and sex. Source: Health Survey for England combined data 2010–12. Joint Health Surveys Unit (Nat Cen Social Research & UCL) 2014

Patients are also classed as having coronary heart disease (CHD) and the likelihood is based primarily on their age, see figure 2.

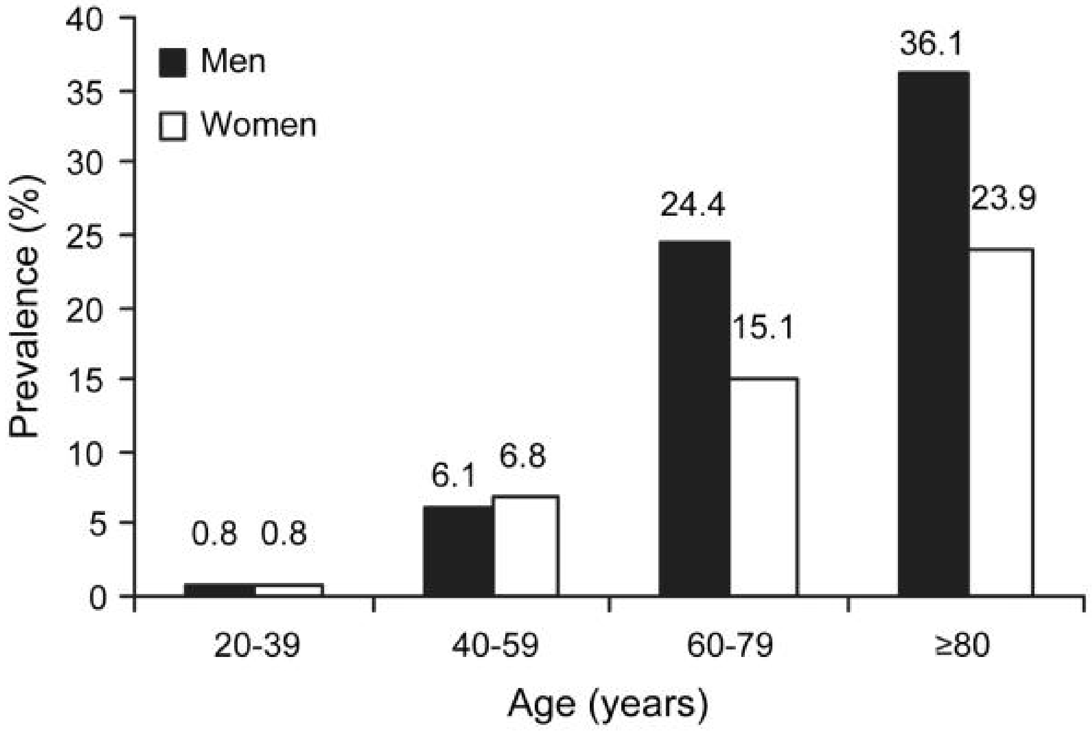


Figure 2: Statistics used to calculate the likelihood of the patient having coronary heart disease. [21]

3.2 Visualisations

Figures 3 and 4 show the distribution of the patients' weight and age, respectively. For weight, I tried to model the worldwide prevalence of obesity, although the rate of obesity varies between countries. In lieu of data about Healthentia users, I used national statistics from European countries. The number of underweight people is probably too high for a developed nation. The mean age of the patients is set to 40, with a good representation of both very old and very young people. The distribution of overweight and obese patients is important given that it directly affects some of the recommendations. The height of patients is also taken into account in the weight distribution, with taller people being on average 700g heavier for every cm above the mean. [22]

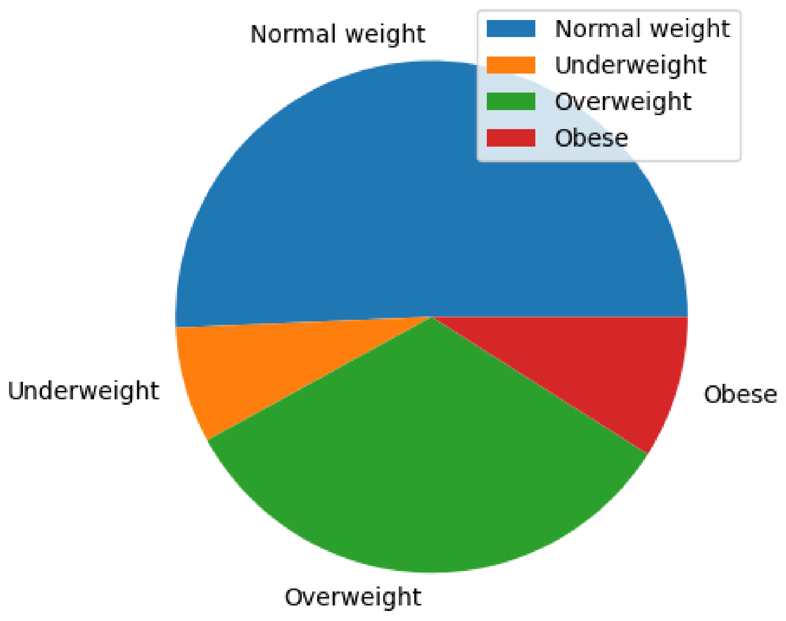


Figure 3: Distribution of overweight and obese patients.

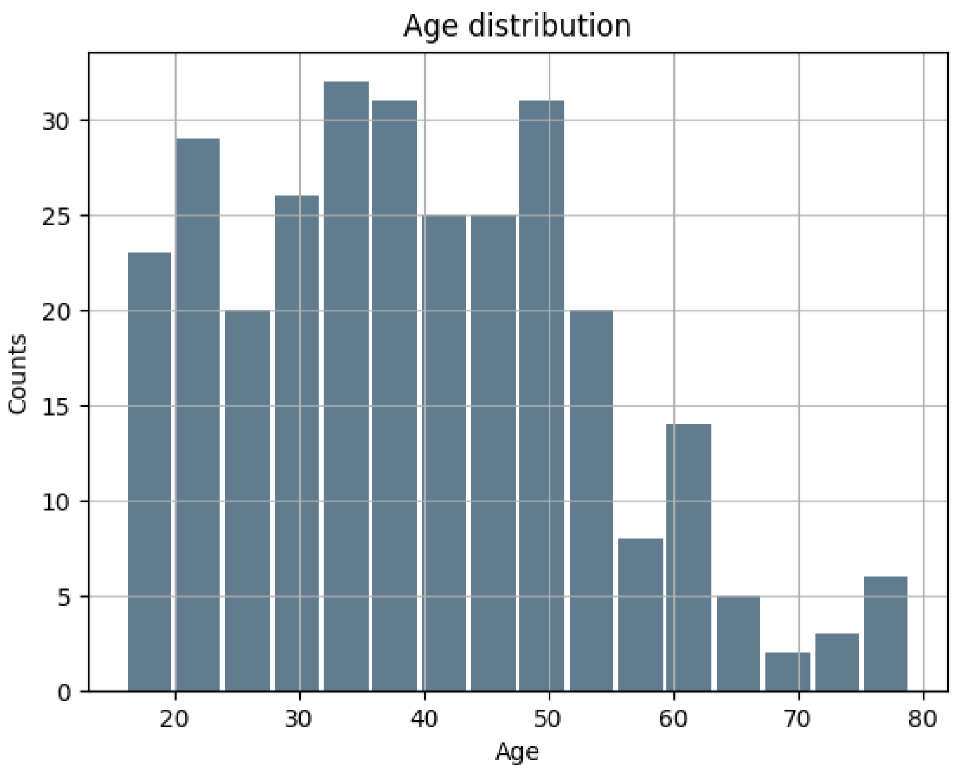


Figure 4: Distribution of patient ages.

3.3 Sleep

For sleep, we generate data about both the duration of sleep in a day, and the *quality* of sleep. The latter is a number between 0 and 1, which is a representation of how well the patient slept. If a patient has sleep apnea, for example, the quality of their sleep will be reduced by 40% (studies show that around 5% of the population suffers from sleep apnea [23]). Anxiety also reduces sleep quality: each point of anxiety reduces sleep quality by 2%. Anxiety is determined by a random draw – some patients are naturally more anxious than others, with values ranging from 1 to 10. If a patient is unemployed, their anxiety is increased by 50%; the proportion of unemployed people is configurable, and the default is 20%. The patient's sleep quality also varies day-to-day, based on factors like how much stamina they had remaining when they finished the day.

Sleep quality is not easy to quantify; the numbers above are simply a representation of a complex reality.

Variables like stamina, health and apnea are important, in that they determine characteristics of the patient, but they are only used internally and not included in the dataset. This is because sleep apnea is often not diagnosed (so the patients don't know they have it). Likewise, characteristics like stamina and health are not known – they have to be inferred from other available data.

3.4 Diet

In terms of diet, we classify the foods the patient eats using the same guidelines used by the Diet Widget of the Healthentia app. The foods are structured in a tree, see the Appendix.

We want to classify the types of foods the patient eats. We are primarily interested in what type of fats the foods contain: it is well-established that certain saturated fatty acids, such as the ones found in meat, butter and palm oil increase Low-Density Lipoprotein (LDL) cholesterol and lower High-Density Lipoprotein (HDL) cholesterol; they also lead to other pathologies like fatty liver disease ([24]). The LDL:HDL ratio is an important because low HDL and high LDL increases the risk of coronary heart disease and stroke ([25]). Additionally, we also want to know whether the patient's diet contains enough fibre, since fibre is known to promote weight loss, increase insulin sensitivity, reduce the chance of developing colon cancer, and benefits heart health ([26]). The consumption of sugar is known to increase triglycerides, promote weight gain, tooth decay, and metabolic diseases ([27]).

For diabetics, the Glycemic Load (GL) of the food is important—foods with a low glycemic load do not lead to large spikes in blood sugar and can help control both hyperglycemia and hypoglycemia. Foods that are low in carbohydrates have a low GL, but even some foods that are mainly starchy – like many fruits, vegetables, pulses and wholegrains – do not have a high GL because their Glycemic Index (GI) is low. These foods are appropriate for diabetics to consume, and are even recommended because of their high fibre content and ability to promote satiety.

We model each patient as being a healthy eater, a random eater, or an unhealthy eater. Healthy eaters are concerned about their diet and will choose to eat healthy foods while avoiding bad foods; unhealthy eaters tend to eat unhealthy foods; and random eaters are indifferent, with a very wide variance in food preferences. Not only do unhealthy eaters eat unhealthy foods, but they are also more likely to eat more, as unhealthy foods tend not to promote satiety as well as healthier foods. This characteristic is modelled as "gluttony" within the simulation, with the values major overeater, minor overeater, and normal eater. The following table shows the mean daily calorie intake of each type of eater. Generally, to maintain a healthy weight, women should eat 2000 kcal/day, and men 2500. The standard deviation equals 10% of the mean, which is based on the literature. [28].

Type of Eater	Women	Men
Normal eater	2000	2500
Minor overeater	2200	2750
Major overeater	2700	3300

3.5 Body Fat Percentage vs BMI

One of the things we did not model in the simulation was whether patients with a high BMI are obese, or muscular; it is simply assumed that patients with a high BMI are obese by definition. In the real world, this is not always the case, with for example body builders. Body fat, on the other hand, is a more reliable indicator of whether a patient is actually obese or not, and it can be combined with BMI to give a more detailed picture of the patient's health. Furthermore, body fat percentage can detect patients who may appear be a of a good weight but have low muscle mass due to a sedentary lifestyle. These patients are at increased risk of infirmity and sarcopenia as they age [29]. Since such patients don't carry out sufficient strength training, the algorithm should still provide them with the correct advice, however. [30]

There are a few reasons for this decision though. Firstly, although body fat percentage can be estimated using cheap devices that rely on bio-electrical impedance, the results are not that reliable [31], and an accurate measure typically requires sophisticated devices like DEXA, MRI or ultrasound (ibid). Body fat percentage is not currently recorded by the Healthentia application. Secondly, the number of people with high BMI but low body fat are a relatively small proportion of the population. Finally, it is possible to workaround the issue by combining BMI data with exercise data: patients who have a high BMI but carry out a lot of strength training can be classified as not obese. For example, Hafthor Björnsson, who played the Mountain in Game of Thrones, weighed 155kg when he fought Eddie Hall [32], and with a height of 200cm, he had a BMI of 38.8 – well into the territory of obesity. Yet his body fat was estimated at just 16.5% [1], which is in the healthy range for men. [33]

Still, in an ideal world, body fat percentage should be simulated/recorded in order to provide a better picture of the patient's health, provided that the veracity of the data is taken into account.

[1]No official data has been published, this data came from a website called BodyWHAT. But since body fat percentage is easy to estimate from pictures, especially by professionals in the fitness industry, this number can be considered reliable.

3.6 Exercise

In terms of exercise, the patients can pick one of the following activities: run, walk, hike, cycling, swimming, gym and dance. They carry out this activity for a duration of time measured in minutes. This time is subdivided into sedentary, light, moderate and very intensive exercise moments. Ideally, we would like to see a patient that carries out a varied mix of exercises.

Different activities train the body in different ways. Core involvement is higher in some activities and less in others: gym training involves a high degree of core exercises (planks, crunches etc.); dance and swimming build core strength to some extent, while hiking does not involve the core muscles very much. The level of core involvement is treated as a percentage, and computed based on the type of exercise carried out by the patient. In terms of strength training, working out at the gym is considered 100% strength exercise; swimming and cycling are treated as 50% strength, 50% cardiovascular; and the other exercises are considered cardiovascular only.

Patients are classed into a number of phenotypes which affect the amount of exercise they carry out. These are: athletic; normal; sleep lover and unfit. Within these phenotypes, additional tweaks can be made, such as those that determine the patient's preference to working out indoors vs outdoors.

We considered simulating other parameters which could possibly be collected by the Healthentia app, such as blood glucose levels, serum cholesterol, and LDL count. However, this proved quite difficult, and I deemed that the time available for the thesis was not sufficient to justify the effort of doing this.

3.7 Scope of Recommendations

There is a wide area of specificity in which the recommendations may fall under. In this section, I clarify which "level of abstraction" the algorithm works on. Think of the recommendations as a tree: we start with raw data at the root, we formulate more specific recommendations as we go down the tree, and we finish with day-to-day advice at the nodes.

Levels of abstraction:

- Level 0: raw patient data. This is the root of the tree.
- Level 1: More specific advice within that area, but still higher-level. So, for example, a level 1 recommendation would be "patient needs to focus on core strength", or "patient needs to focus on reducing intake of high GL foods". This type of recommendation is memory-less. Formally, we treat this as being a multi-label classification problem.

- Level 2: very specific, day-to-day advice, e.g. "eat some oats instead of sugary cereals this morning" or "plank for 1 minute this afternoon". This advice can change regularly based on the patient characteristics and past responses, so if a patient doesn't like doing planks, the algorithm can suggest crunches instead. This type of recommendation requires the algorithm to "remember" patient responses. Giving recommendations at this level may involve analysing time series: for example, we want to analyse the trend in the patient's daily exercise activity, and if it is statistically significant, we can congratulate them for doing more exercise.

For my thesis, I focused on level 1. The Healthentia "virtual doctor" gives L1 advice, whereas the Healthentia "coaches" give L2 advice. So, for example, the virtual doctor can give 3 recommendations, two of which are related to diet and one which is sleep-related. The patient will then be asked to interact with the diet coach and the sleep coach. If the patient disagrees with the recommendations provided by the virtual doctor, they can change it.

The L1 advice changes over long periods of time, typically months; the L2 advice changes over days or weeks. The L2 advice needs to be evaluated for medical accuracy; it has to be appropriate for the patient, and it also has to prioritise the more important advice. It is personalised in the sense that it uses patient data, and the advice changes at different points in time (months). But it is not personalised to the same extent as L2 advice is; the Healthentia coaches are designed to actually have a kind of conversation with the user. See 5 for a diagrammatic representation.

The main advantage of decoupling L1 from L2 advice is that different approaches can be used. For L1 advice, it is possible to use machine learning by treating the problem as one of multi-label classification: each patient is labelled with 1–5 recommendations, and an algorithm is trained on the labelled dataset. The L1 advice generated by my algorithm can then be used as input for a virtual coaching algorithm that will be developed by my company at a later date.

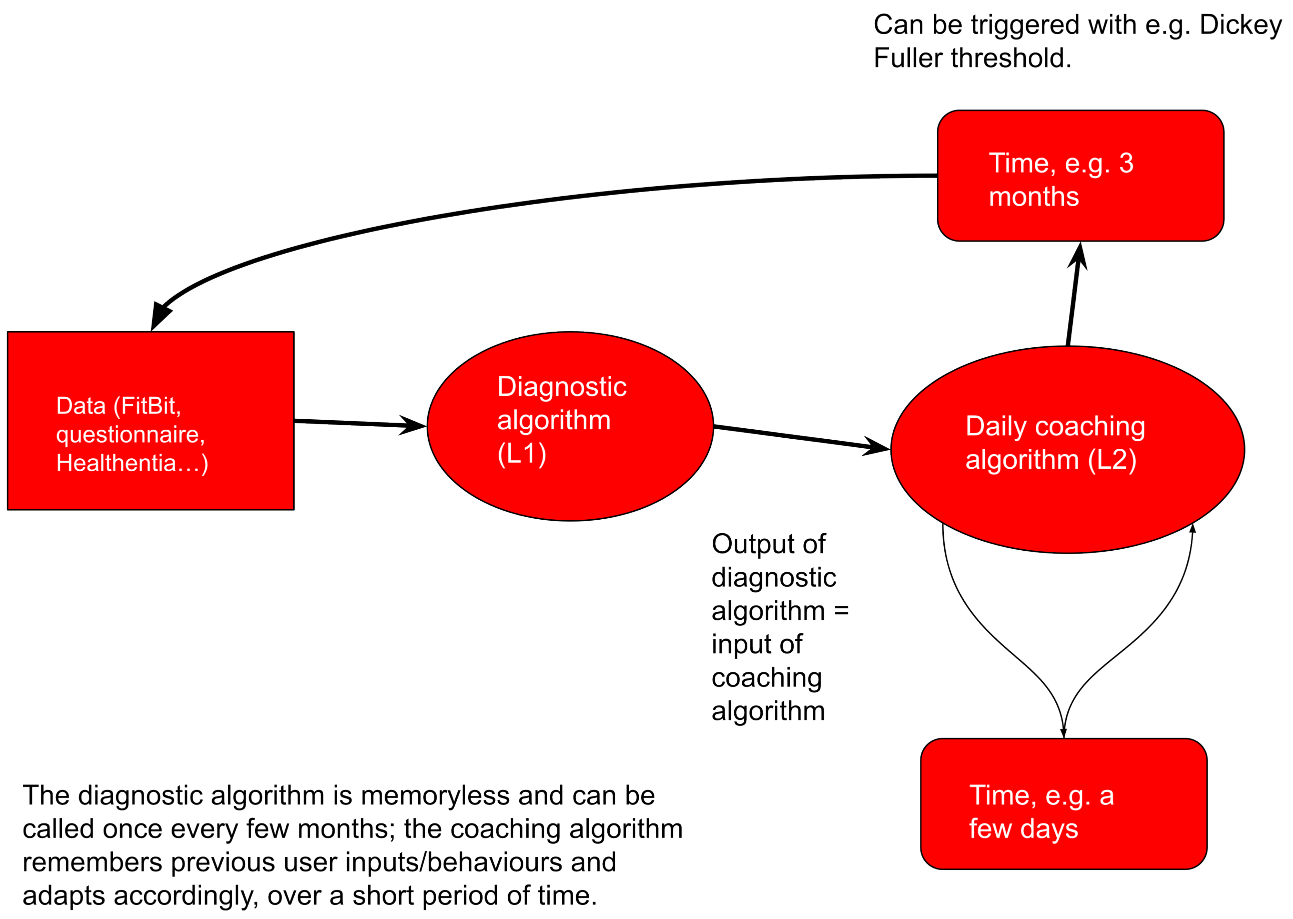

Can be triggered with e.g. Dickey Fuller threshold.
Time, e.g. 3 months
Data (FitBit, questionnaire, Healthentia…)
Diagnostic algorithm (L1)
Daily coaching algorithm (L2)
Output of diagnostic algorithm = input of coaching algorithm
Time, e.g. a few days
The diagnostic algorithm is memoryless and can be called once every few months; the coaching algorithm remembers previous user inputs/behaviours and adapts accordingly, over a short period of time.

Figure 5: Data flow diagram of L1 and L2 algorithms.

Note that positive re-enforcement/support falls under L2 recommendations and won't be included in my list of L1 recommendations. This is not to say that positive re-enforcement isn't important, as research suggests it is. For example, Appel et al. [34] found that patients who received support lost significantly more weight than those who did not. Likewise, Matteson [35] found that positive re-enforcement increased the amount of exercise that elderly patients carried out.

3.8 List of Recommendations

1. (Exercise) Patient should do more cardiovascular exercise.

2. (Exercise) Patient needs to focus on core strength.
3. (Exercise) The patient needs to do more resistance training.
4. (Exercise) The patient needs to do more intense exercise.
5. (Diet) Patient should reduce their intake of unhealthy saturated fats.
6. (Diet) The patient should eat fewer sugary foods and drinks.
7. (Diet) The patient needs to eat foods with a low GL index and high fibre content.
8. (Diet) Portion sizes need to come down.
9. (Sleep) The patient has to focus on getting more sleep.
10. (Sleep) The patient needs to improve the quality of their sleep.
11. Patient is healthy, does not need to do anything.

In total there are 11 possible recommendations from which the algorithm must pick at least one and maximum 5. We initially chose a maximum of 3 recommendations, because psychology research shows that people can only work on a limited number of goals at a time (see [36] and [37]) – giving the patient too many recommendation would overwhelm them. However, it was discovered that some patients needed more than 3 labels; if the limit of 3 were enforced, it would make it difficult for a machine learning algorithm to learn, since some labels would be chosen arbitrarily. We decided to extend the limit to 5 when the patient had multiple values that were very poor. See the labelling section for more information.

We only intend to give the patient at most 3 recommendations at a time in the Healthentia app. The idea is that we give the patient time to focus on the first three recommendations, and record their progress over time; if we are satisfied that real changes have been made, we can call the algorithm at a later point in time to re-evaluate the patient's situation. The patient would then focus on the remaining 2 recommendations, or perhaps different ones will be needed. A person's lifestyle habits naturally change over time anyway, so the recommendations given by this algorithm only represent a snapshot.

3.9 Labelling of the data

Labelling was carried out by multiple coders using Google Docs. The data was colour-coded into red, green and amber, based on whether the patient met the official guidelines published by public health authorities.

Parameter	Men	Women
Saturated fats	Less than 10% of energy intake	Same
Fibre	Minimum 31g per day	Minimum 25g
Sugar	Maximum 31g per day	Maximum 25g
Exercise	At least 150 minutes per week	Same
BMI	over 30 is obese; over 25 overweight	Same

For strength training, less than 50 minutes per week was labelled red; values between 50 and 75 were labelled amber; and values greater than 75 were labelled green. This is computed as the product of proportion and duration.

For core exercise, the lower bound was 60 minutes per week (red), and the upper bound 75 minutes (green), with amber for values in between.

The exact guidelines for labelling are included in the appendix. We had to be strict about the labelling to ensure consistency. For example, we specified that a patient could only receive 4 or 5 recommendations for values that were labelled red, not amber. In hindsight, this was too strict, as a number of patients had many amber variables, and it was difficult to choose.

3.10 Ranking of recommendations

We discussed the possibility of ranking the recommendations in order of importance. But this idea was rejected because it was hard to agree on which recommendations should be considered more important than others. For example, if a patient only sleeps 7 hours a day but is obese, can we say that increasing sleep is more important than cutting down on calories? Furthermore, it is unlikely that we would obtain good inter-annotator agreement, as we would be testing not only which recommendations, and how many, but also in what order.

It should be noted that there is an implicit ranking of recommendations in that not all recommendations are chosen if there are more than 3 amber values, or more than 5 red.

Another possibility is to split the recommendations into two sets, with very important recommendations in the first set, and less important recommendations in the second set; this would generally correspond to whether the respective values were coded red or amber. The order within these two sets would not matter. I think this would be an improvement over the present solution, but due to time concerns, it was not implemented (as a large number of patients would have to re-labelled this way). It would also significantly complicate the machine learning task, as it would become a multi-output problem.

3.11 Visualisation of labels

We can see that the data is very unbalanced. Recommendations 6, 7 and 9 are very frequent, which makes sense as a large percentage of the population overeat sugar, do not enough fibre and do not sleep sufficiently. Label 2 is completely unused, which may be a consequence of how we modelled exercise in the simulation, as all the patients who exercised did some amount of core training. This is not necessarily realistic. Label 4 was only used once, though again this does not necessarily reflect real data. Recommendation 11 is unsurprisingly rarely used, since few patients are perfect.

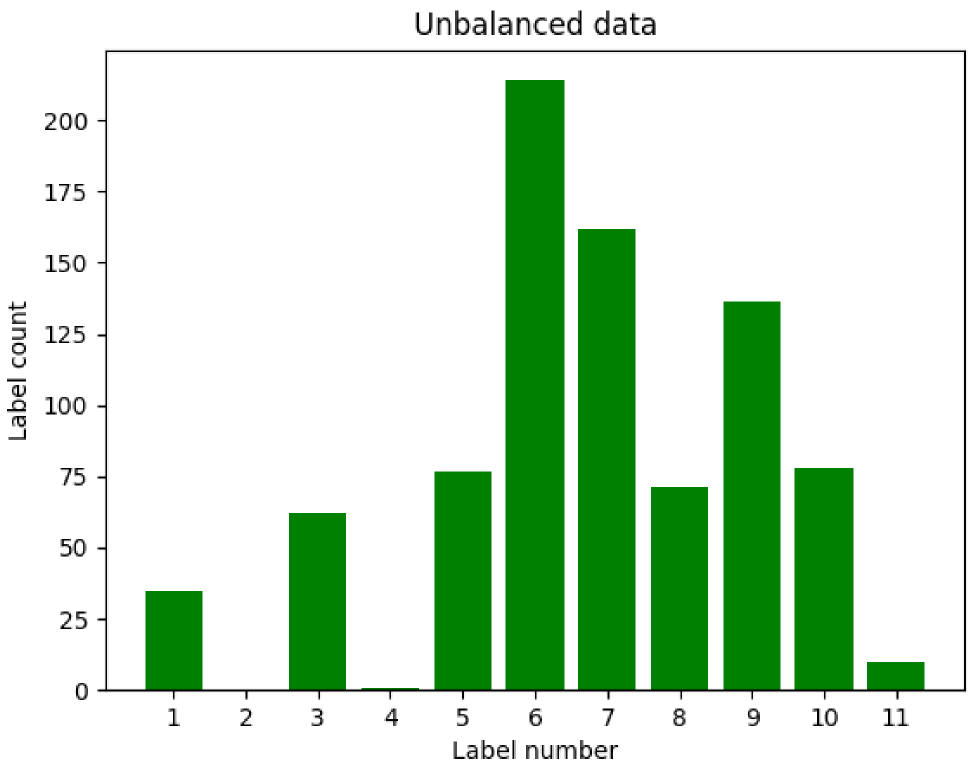


Figure 6: Frequencies of the labels

3.12 Model Selection

A multi-label classification problem is inherently different from a standard multi-class or binary classification problem in that the labels are not mutually exclusive. This makes it challenging because most types of machine learning algorithms were designed with a simple classification problem in mind. There are a few different ways to handle this.

The first is to use an algorithm that can is inherently capable of handling multi-label data; a neural network with a sigmoid activation function in the output layer is perfect for this, as each neuron corresponds to a label, and the neurons with the highest probability can be chosen. For example, if neurons 3, 5, and 7 in the output layer have probabilities 0.9, 0.75 and 0.7 respectively, while the other neurons have probabilities below 0.5, we know that 3, 5 and 7 are the best recommendations (and we are very sure about number 3). If 5 neurons have a high probability, then we know the patient might need 5 recommendations. If only one neuron has a high probability, we give them only one recommendation, and so on. Thus, the neural network gives as an idea about the *certainty* of a prediction, in contrast to some other algorithms like decision trees or SVM.

An adapted algorithm is where an algorithm that is not usually capable of multi-label classification is adapted to do this. One such implementation is a K-nearest-neighbours algorithm proposed by Zhang et al. named `MLKNN`. [15]. It uses the Maximum A Posteriori (MAP) principle in order to take into account correlations between labels. It does this by estimating prior and posterior probabilities from the training set (based on frequency).

The second approach is called binary relevance, and the idea is to turn each label into a binary classification problem. So if we have a list of 11 0/1 encoded variables, we classify each one individually. This approach has the advantage of being compatible with a wide range of algorithms, such as Support Vector Machine (SVM), logistic regression, decison trees and so on. However, this approach is somewhat naive in that it does not consider correlations between labels. For example, we know that labels 5, 6, and 7 are correlated because we model unhealthy eaters as eating poorly in all three areas. In this category, KNN, decision trees and SVM were used.

For KNN, the BRKNNClassifier was used. In general, KNN also has the advantage of giving us some idea of the certainty of a prediction because it allows us to inspect the distances between points, unlike most decision trees, although it is still a discriminative algorithm. For decision trees, I used the `DecisionTree-Classifier` of sklearn. A random forest classifier was not considered because, although it is less likely to overfit, it is harder to visualise and understand compared to a simple decision tree. As we see in the results section, overfitting is not a problem if a bit of pruning is done.

SVM is a model that generally performs very well on binary classification problems. It is similar to a linear perceptron classifier in that it splits classes using a straight line, the difference being that SVM calculates the line that is the midpoint between the two classes. It would probably perform well on labels 1 and 9, since it is easy to determine whether those labels are relevant based on whether the patient does enough exercise and gets enough sleep. However, SVM is not designed for multi-label classification, and it cannot take into account relations between labels, e.g. if a patient does not do enough exercise in general, labels 2, 3, and 4 do not apply.

A third approach is called Label Powerset, and it transforms the problem into a mutually exclusive classification problem where each class represents a combination of labels. In theory, there are a large number of possible combinations since we have 11 labels and up to 5 recommendations. But in practice, there are only 62 combinations present in the dataset. This is because of a few different reasons. Firstly, recommendation 11 is mutually exclusive with the others. Secondly, we don't have duplicate recommendations, and thirdly, we are indifferent to the order of recommendations. We also know that recommendations 2 and 4 are very rarely used. However, this approach would not generalise well to novel data because a patient may require combinations of recommendations that were not present in our training set. Therefore I decided **not** to use it.

What characteristics are desirable in a machine learning algorithm, in the context of our problem? I have already mentioned quantifying uncertainty: we want an algorithm that gives us an idea of the sureness of a prediction. Neural networks, logistic regression and to some extent KNN are good at this, whereas SVM and decision trees are not. A second characteristic of a good algorithm is that it is understandable: we can explain how it works to a doctor. KNN and decision trees are relatively easy to understand, whereas neural network are notoriously difficult, as the weights have no intuitive meaning, and a well-performing neural network may have a lot of neurons. A third characteristic is that we want an algorithm with few parameters, applying the principle of Occam's Razor. A nearest-neighbour algorithm might have one or two parameters, whereas a neural network has several, and the possible combinations of neurons are theoretically infinite, limited only by computational resources.

3.13 K-fold Cross Validation

An 80:20 split was chosen for the train:test ratio, and k was chosen to be 5. Since we have 300 samples, that means 240 datapoints are used for training, 60 are used for testing, and the process is repeated 5 times to calculate the mean and variance. The assignment between training and testing is randomised.

3.14 Scaling

Many machine learning algorithms require the data to be scaled in order to calculate distances, otherwise variables with a larger scale (e.g. 1–100 instead of 0 and

1) will skew the predictions. For the KNN and SVM algorithms, normalisation was used:

$$y = \frac{x - min}{(max - min)} \tag{1}$$

For the neural network, using standardisation gave better results. I think this is because standardisation leads to a smaller gradient than normalisation, so it is easier for the algorithm to converge using gradient descent.

$$y = \frac{x - \mu}{\sigma} \tag{2}$$

Standardisation assumes that the data is approximately normally distributed, which is not always the case. For example, gender is either 0 or 1 with nothing in between. I generally prefer normalisation for this reason (standardisation is intended to be used by algorithms that assume a normal distribution, like logistic regression). But the algorithms proved surprisingly sensitive to what kind of scaling was done, so I opted for the type of scaling that gave the best performance in order to provide as fair a comparison as possible.

The decision tree does not require standardisation so none was performed.

3.15 Hyperparameter Optimisation

KNN: MLKNN has two parameters: k, the number of points to use for distance calculation and voting; and s, which controls is a smoothing parameter that determines the strength of the uniform prior (the default, 1, corresponds to Laplace smoothing). The third parameter is the type of distance metric, be it Minkowski (the default), Manhattan or Euclidean. The k parameter is the most important. K is important because using too small a value for k will result in unstable decision boundaries, whereas too high a k leads to too much smoothing. A good rule-of-thumb is the root of n training samples, which is 15 in our case. Therefore I picked a range of k from 10 to 20. For the smoothing parameter, I chose the range 0.5, 0.7 and 1. As MLKNN is not very computationally expensive, optimisation was carried out using `GridSearchCV` from sklearn, which tests all possible combinations. This resulted in k=13 and s=0.7 as the best parameters.

Decision Tree: For the decision tree, Gini was used as the impurity metric, and pruning was done using minimal cost complexity. Initially, I chose `ccp alpha` parameter as 0.03, because it seems to give the best trade-off between test and train accuracy, see figure 7.

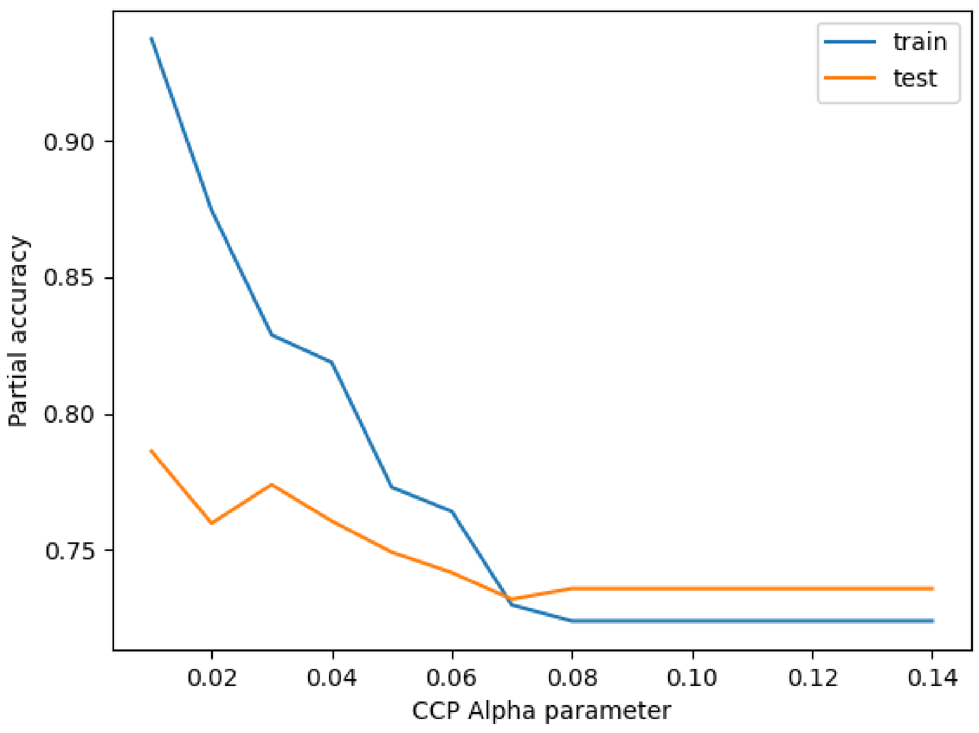


Figure 7: Train vs test accuracy for the decision tree

However, the decision tree had very poor accuracy on label 11, and further investigation revealed that there were no splits for label 11 because the tree had been pruned too aggressively. Reducing the alpha parameter to 0.01 increased performance both on this label, and across the board, since the tree was too conservative in giving recommendations. This came at the cost of slightly increased variance, but this was a worthwhile tradeoff given the significant improvement in performance.

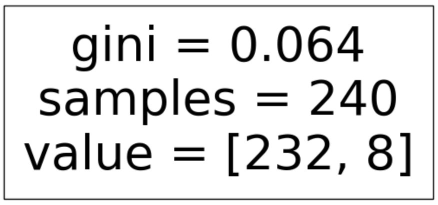


Figure 8: Label 11 split with ccp alpha=0.03

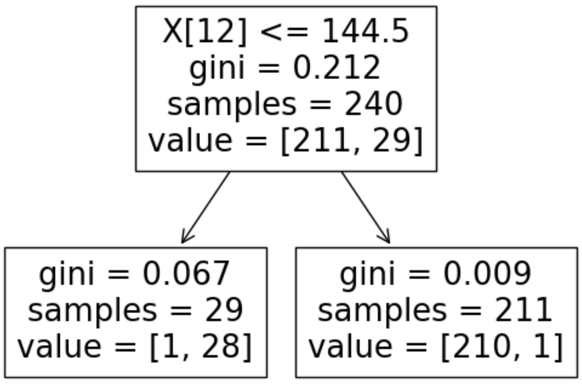


Figure 9: Label 11 split with ccp alpha=0.01

SVM: Hyperparameter optimisation was carried out in the same manner as for KNN, using `GridsearchCV`. Linear, poly and rbf kernels were optimised alongside the C parameter. The best C parameter was 0.5 and the polynomial kernel performed the best - this suggests that the higher-dimensional data is not linearly separable.

Neural Network: As neural networks have a large number of hyperparameters, with a virtually infinite combination of neurons in the hidden layers, it is essential that hyperparameter optimisation is carried out. Choosing the wrong hyperparameters can have a big impact on model performance. However, neural networks are computationally expensive to train, especially if there are a lot of neurons, and it is often not viable to iterate over the entire search space using `GridSearchCV`.

Some hyperparameters are known a priori, though:

- Input neurons: 17, which corresponds to the number of predictors.
- Neurons in the output layer: 11, which corresponds to the number of labels.
- Activation function of the output layer: sigmoid. Sigmoid is the perfect function for representing probabilities, since the output will always be between 0 and 1. Softmax is inappropriate for multi-label classification because it assumes the classes are mutually exclusive, leading to problems. For example, when multiple classes are true, the confidence interval is incorrectly split

between them. When no classes are true, softmax picks a winner anyway. [38]

- Activation function of the hidden layers: ReLU is preferred when there are many hidden layers because of the vanishing/exploding gradient problem [39] so this is what I went with. In short, the exploding gradient problem occurs during backpropagation when multiplying many derivatives that are large, leading to an overflow error. The vanishing gradient problem happens when the product of the derivatives becomes zero, so that the model is no longer able to update its weights and learn.

The choice of solver should be considered carefully. We know that the adam solver (the default in sklearn) is inappropriate for small datasets because it converges slowly, like most stochastic gradient descent methods – it was designed to work on large datasets with the idea of being fast. The lbfgs solver is probably best because although it is computationally expensive on a per-iteration basis, it converges faster, and is therefore perfect for small training sets. Stochastic gradient descent might also perform well if the learning rate is chosen appropriately. If the learning rate is too low, the neural network will not converge; if the learning rate is too fast, the neural network will "overshoot" (fail to converge to the right value). It is best to start with a higher learning rate and gradually reduce the rate during each step. This can be accomplished by setting the learning rate to inverse scaling, and finding the right values for `learning_rate_init` and `power_t`.

A different option is to use the lbfgs solver. This generally converges faster than stochastic gradient descent and has fewer parameters to optimise. However, I discovered that it does not perform any better than stochastic gradient descent for our dataset, see figure 11.

Two other parameters that need to be considered when using SGD are momentum and the mini-batch size. Momentum is used to smooth out noisy data by taking the mean of past values. The default value of 0.9 takes the mean of 10 points, which has empirically been shown to work well ([40]), though a reasonable range should be considered. The mini-batch size determines how many points to sample for the stochastic gradient descent. Like momentum, using too many points leads to much smoothing (and thus a failure to reach the global optimum; the algorithm gets stuck at a local optimum), whereas using too few points leads too much noise.

The alpha parameter determines the amount of L2 regularisation: the idea is to penalise large weights to prevent overfitting. Too large a value results in underfitting, too small a value leads to overfitting.

The first approach is to optimise the hyperparameters using a genetic algorithm. Of the available libraries, `PyGAD` does not support multi-label classification at the time of writing, but `GASearchCV` from sklearn-genetic-opt does. There are two criteria a problem must satisfy to it suitable for a genetic algorithm:

1. Clear and fast way to evaluate fitness: we can use any performance metric like Hamming Loss or partial accuracy to minimise or maximise, respectively. These functions are very fast to compute, especially with such a small training set.

2. Search space is well-constrained: we have a finite number of parameters with a finite range for each. The biggest uncertainty is around the number of layers and how many neurons should be in each layer, but we can use a reasonable range.

The idea behind a genetic algorithm is to create a population, keep a number of elites in the next generation, and combine "genes" from the best-performing "parents". This is known as cross-over. Some features can also be changed randomly; this is called mutation. Generally speaking, genetic algorithms are adept at obtaining a "good enough" solution in a reasonable amount of time, just like evolution in the real world. There is no mathematical guarantee that a genetic algorithm will converge to the "best" (global optimum) solution – but they perform well in practice.

A different approach to hyperparameter optimisation is to use so-called sequential model-based optimisation, which is implemented by `BayesSearchCV` and various other libraries. The idea is that if the objective function (the neural network, in other words) is expensive to compute, a surrogate function – an approximation – is used instead. Over multiple iterations, the surrogate function is updated to more and more closely match the real function. The algorithm does this by remembering previous hyperparameter:score pairs. A "selection function" is then used to intelligently choose the next hyperparameters, which is done by maximising the expected improvement function.

Yet another approach proposed in the literature is to use gradient-based optimisation for hyperparameters. But gradient-based methods perform well when the function is smooth and convex, and hyperparameters are neither of those things [41]. For example, deciding between L1 and L2 regularisation is non-smooth.

In the end I chose to use a genetic algorithm for optimisation. The results are presented in figure 10. Note that cross-validation is used internally to compute the fitness function for each individual neural network.

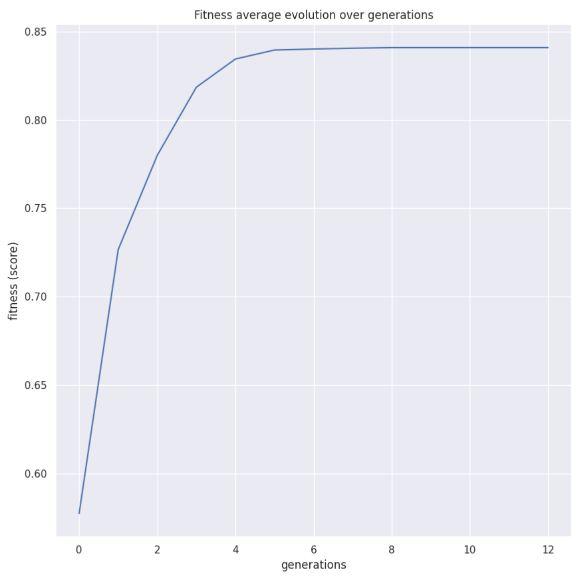


Figure 10: Performance of the SGD-based neural network during each generation.

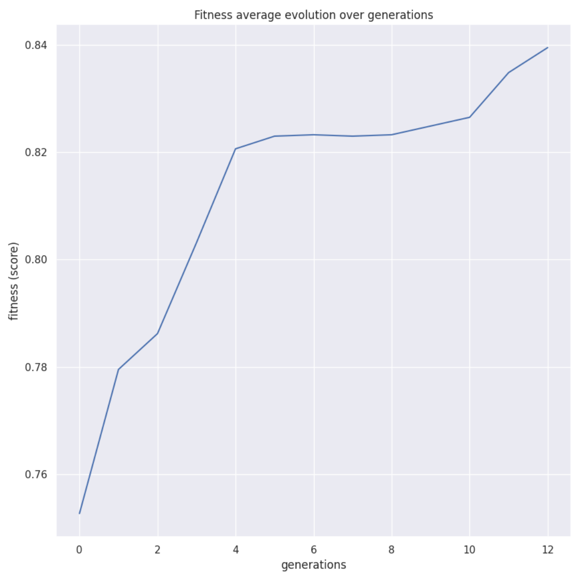


Figure 11: Performance of the lbfgs-based neural network during each generation.

The optimal parameters for the SGD neural network are:

Table 1: Optimal hyperparameters

learning rate init	0.275
hidden layer sizes	102
power t	0.413
momentum	0.917

Interestingly, using a deeper neural network (up to 10 layers) did not improve performance, as the best-performing one has a single hidden layer. Nor did increasing the number of neurons lead to better performance.

4 Outcome Measures

The purpose of this section is to describe which outcome measures were chosen to evaluate the performance of the different algorithms, and why.

4.1 Inter-annotator metrics

Krippendorff's alpha was chosen initially as the best measure of inter-annotator agreement for multiply labelled data (since Cohen's kappa does not support multiple labels). Krippendorff's alpha is a number between 0 and 1 where 1 represents perfect agreement and 0 represents agreement that is no higher than what would be expected **by chance**. This is an important distinction. Krippendorff's alpha can be low even when raw measures of agreement are high – especially if the dataset is small, or the labels are unbalanced. For example, in a binary classification problem where 98% of 100 patients are labelled 0 (no cancer) by one coder, and the other coder labels 99% with no cancer, the alpha score can be quite low (sensitivity would be a better metric to use). Thus, in certain situations, Krippendorff's alpha may not be appropriate.

The implementation provided by the `nltk` package was used, as it had the best documentation compared to the other Python libraries (namely `krippendorff` and `disagree`). MASI was chosen as the distance metric, although Jaccard distance would be an equally valid choice. The choice of implementation should not be treated as a mere technicality: different implementations gave totally different results. The lack of documentation specifying how the input should be formatted likely led to GIGO.

4.2 Strict accuracy

Strict accuracy, also known as zero-one loss, refers to the proportion of predictions that perfectly matches the target output. For example, if the target output is {6, 7, 8} and the prediction is also {6, 7, 8} the score is 1 for that patient. However, if the prediction differs even a little, say {5, 6, 7} then the score is 0. This is a very strict measure that does not accurately represent performance in a multi-label scenario [42]. However, it is easy to implement and has an intuitive meaning.

4.3 Hamming Loss

In addition strict accuracy, Hamming Loss was computed using the `sklearn.metrics` library. Hamming Loss represents the number of substitutions required to make two sets match, with matching sets requiring 0 substitutions. Hamming Loss is a better measure of agreement than zero-one loss in our situation because there are a lot of instances where the recommendations have a partial match. For example, the recommendations {5, 6, 7} and {1, 6, 7} have two correct matches. However, Hamming Loss is not very intuitive because a lower number is better. This makes it hard to interpret.

4.4 Partial accuracy

Because Hamming Loss is difficult to interpret, I coded my own performance measure specifically for multiply labelled data. It is named partial accuracy, and it calculates the average percentage of matching recommendations between two coders. So if coder 1 gives a patient the recommendations {5, 6, 7} while coder 2 gives {1, 6, 7}, the score will be 0.66. It also works if the number of recommendations is different. For example, if coder 1 gives recommendation 10 while coder 2 gives {1, 5, 7, 9, 10}, the score will be 0.2. Thus, this metric penalises greedy algorithms that give too many recommendations. If the recommendations match perfectly, the score is 1, as expected, and 0 if no recommendations match. This function is commutative; it does not assume a "ground truth", so it is also an appropriate measure of inter-annotator agreement as well as model performance.

4.5 Label accuracy

We are not only interested in knowing a model's performance in the aggregate; we also want a breakdown of its performance on individual labels. I therefore coded a measure of label accuracy, where each label is assigned a score from 0 to 1.

Note that unlike partial accuracy, label accuracy is **not** commutative; it assumes a ground-truth.

4.6 Length Mismatch & Length Ratio

Length mismatch is simply the number of instances where predicted sets do not have the same length as the target output, divided by the total number. A value closer to 0 indicates better performance. Note that the length mismatch function does not care whether the predictions actually match the targets; it is simply a measure of whether the algorithm is giving the right number of recommendations.

The length ratio can be interpreted in conjunction with the length mismatch ratio to tell us whether an algorithm is giving too few or too many recommendations. A ratio above 1 indicates that an algorithm has a tendency to give the patient too many recommendations; a ratio below 1 indicates too few. If the ratio is 1, this does **not** mean that the algorithm gives the right number of recommendations – it just means that the tendency goes either way, with too many or too few recommendations depending on the patient. Therefore, these two numbers only have meaning when taken together.

4.7 Confusion Matrix

A multi-label confusion matrix works just the same way as it does for binary classification; each individual label can be thought of as a separate binary classification. It is also possible to calculate precision and recall for each label in this way, although this is left as an exercise for the reader.

5 Results

The purpose of this section is to report on the results of all the outcome measures aforementioned.

5.1 Interannotator reliability

Me and two of my colleagues at Innovation Sprint labelled the first 50 patients of the dataset independently. The Krippendorff's alpha statistic was calculated using a 3-way comparison; the other statistics are the means of pairwise comparisons. The results are summarised in table 2 rounded to two significant figures.

Table 2: inter-annotator reliability

Metric	Score
Krippendorff's alpha	0.64
Hamming Loss	0.067
Strict accuracy (zero-one loss)	0.61
Partial accuracy	0.86

We see that 61% of the recommendations match perfectly and 86% of the sub-recommendations match, which we consider a good result. The Hamming Loss is quite low and Krippendorff's alpha is significantly better than chance. In particular, the partial accuracy score serves as a good intuitive benchmark; it also gives us an upper bound for how well an algorithm can perform.

We think that if 4 or 5 recommendations could be given more freely, the inter-annotator metrics would improve – this is because it's difficult to choose only 3 recommendations when the patient has numerous issues that need fixing. This is something we would change in the future.

5.2 Partial accuracy

Table 3 summarises the partial accuracy scores for each algorithm. The knowledge-based algorithm performs the best. The neural network and SVM tie with the decision tree, but they have lower variance, meaning they perform more consistently than the decision tree. The difference between MLKNN and BRKNN is marginal: both perform poorly.

Table 3: Partial accuracy results (using cross-validation)

Algorithm	Mean Partial accuracy	Standard Deviation
MLkNN	0.72	0.015
BRkNN	0.71	0.017
Neural Network	0.83	0.014
Decision tree (a=0.03)	0.78	0.024
Decision tree (a=0.01)	0.83	0.027
Knowledge algorithm	0.87	NA
SVM	0.83	0.012

5.3 Strict Accuracy

We see the same story repeated with the strict accuracy metric. The KNN algorithms perform poorly; the other ML algorithms perform about the same (though the decision tree has higher variance). The knowledge-based algorithm still performs the best by quite a margin.

Table 4: Strict Accuracy

Algorithm	Mean Strict Accuracy	Standard deviation
MLkNN	0.33	0.038
BRkNN	0.35	0.034
Neural network	0.53	0.036
Decision tree	0.42	0.048
Decision tree (a=0.01)	0.52	0.049
Knowledge algorithm	0.65	NA
SVM	0.54	0.025

5.4 By Label

Recall that we are not only interested in comparing the algorithms in the aggregate; we would also like to see a breakdown by individual label, as this gives an idea about the consistency of the algorithm's performance. We can observe that MLKNN performs very inconsistently, see figure 12. The KNN algorithm exhibits not only a large variance between labels, but also a large variance in the cross-validation scores, which are represented by the error bars. The neural network (figure 15) and SVM (figure 13) perform the most consistently. The decision tree (figure 14) falls somewhere in the middle. It still struggles with label 11.

The knowledge algorithm is similarly consistent, although it does not perform so well on predicting label 10, referring to sleep quality. It is the only algorithm to have predicted label 4, of which there is a single instance in the entire training set. The advantage of an algorithm that does not rely on machine learning is that it can predict labels which come up rarely in a dataset.

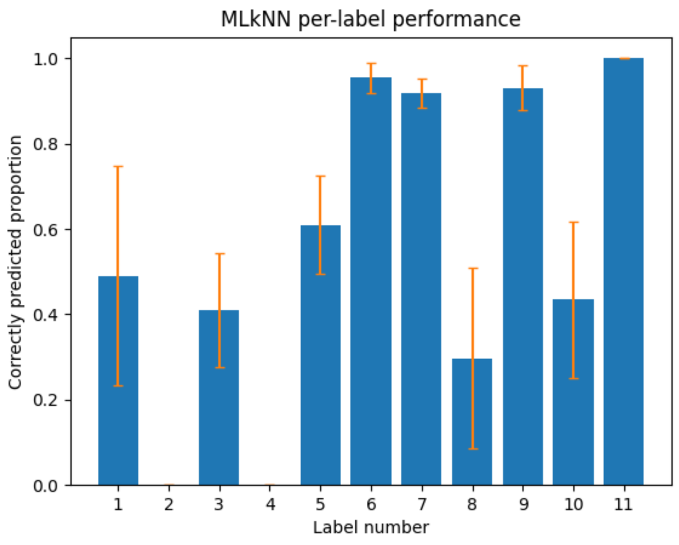


Figure 12: MLkNN performance on each label.

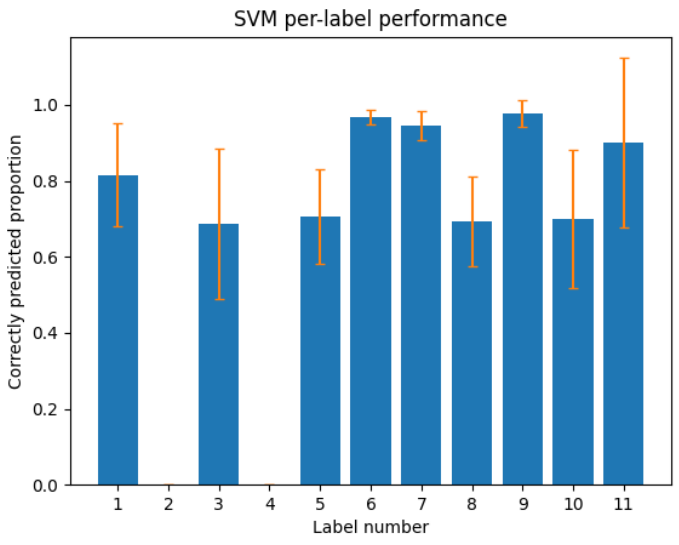


Figure 13: SVM performance on each label.

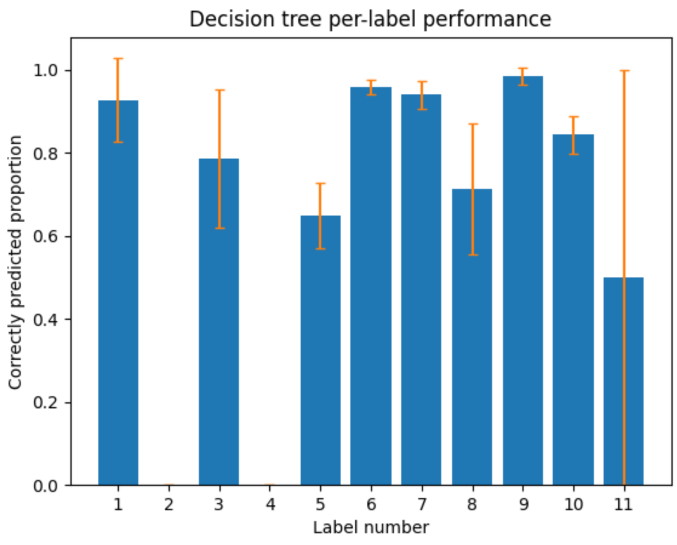


Figure 14: Decision tree performance on each label.

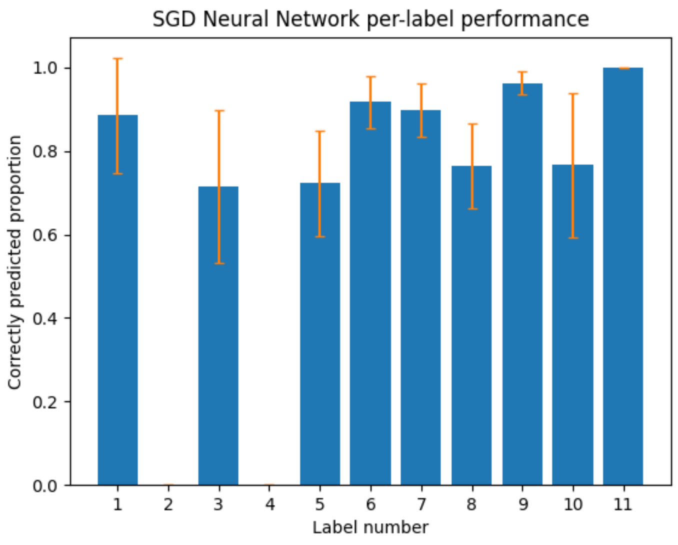


Figure 15: Neural network performance on each label.

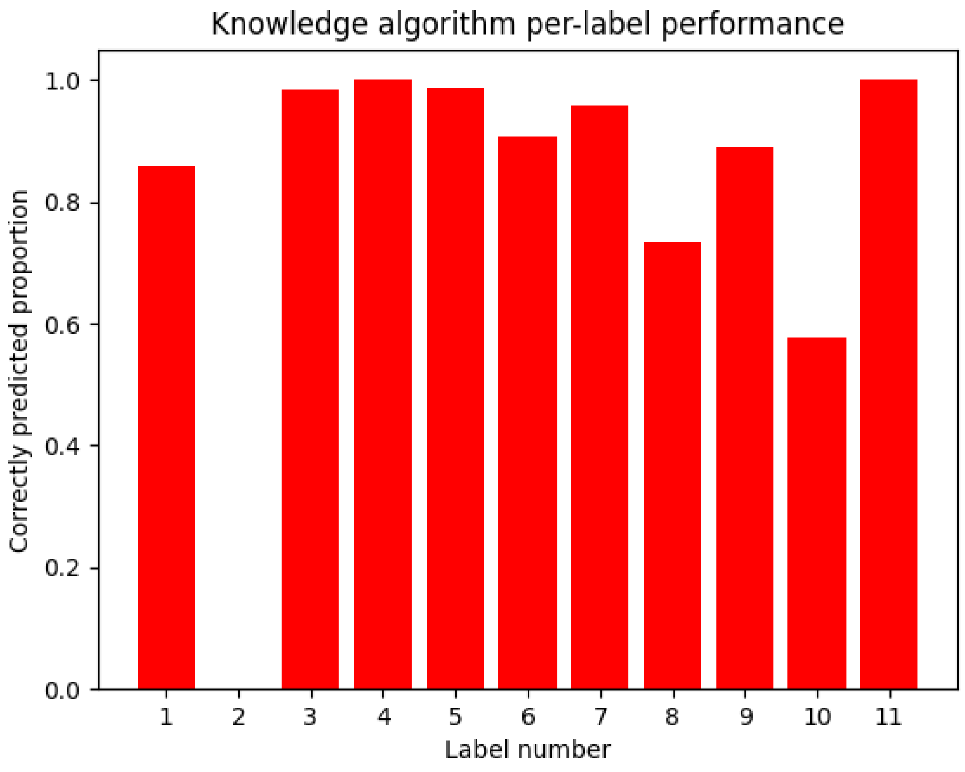


Figure 16: Knowledge algorithm performance on each label.

5.5 Length Mismatch and Length Ratio

The table below summarises the proportion of recommendations that do not match in length. The knowledge algorithm performs much better in this regard than any of the ML algorithms, which really struggle with this. Note that for the ML algorithms, cross-validation was used to compute these scores.

Table 5: Length mismatch

Algorithm	Length Mismatch Proportion
MLkNN	0.47
BRkNN	0.43
Neural network	0.34
Decision tree	0.49
Decision tree (a=0.01)	0.36
Knowledge algorithm	0.06
SVM	0.31

Looking at length ratio, we can see that the KNN algorithms do not give enough recommendations, which partly explains their poor performance.

Table 6: Length ratio

Algorithm	Length Ratio
MLkNN	0.909
BRkNN	0.845
Neural network	1.004
Decision tree	0.959
Decision tree (a=0.01)	0.995
Knowledge algorithm	1.01
SVM	0.995

5.6 Confusion Matrices

It is not practical to report 66 confusion matrices to account for every label and every algorithm. Therefore, label 11 was chosen as the most interesting, since it represents the absence of recommendations. Moreover, a large variance was observed between the different algorithms on this particular label. The results were computed with a train-test split of 80:20 and are reproducible.

Table 7: Neural Network Confusion Matrix

TN = 57	FP = 1
FN = 0	TP = 2

Table 8: MLKNN Confusion Matrix

TN = 57	FP = 1
FN = 0	TP = 2

Table 9: DT Confusion Matrix

TN = 57	FP = 1
FN = 2	TP = 0

The decision tree and SVM do not detect any of the true positives for this particular train:test split. The knowledge algorithm performs the best on this label, with no false positives or false negatives – unlike the neural network, which does have a false positive. This is not a distinction that can be observed simply by looking at the label accuracy.

Table 10: Knowledge Algorithm Confusion Matrix

TN = 58	FP = 0
FN = 0	TP = 2

Table 11: SVM Confusion Matrix

TN = 58	FP = 0
FN = 2	TP = 0

5.7 Applicability to different human coders

The training set was labelled independently by a co-worker at my company. I ran the algorithms with the same hyperparameters on this dataset, in order to see how much of an impact a different human coder makes. The results are presented in table 13. We can see that the results are somewhat lower overall, especially for the knowledge algorithm and neural network. From this I conclude that the human coder does make a difference, and ensuring high levels of inter-annotator agreement is important.

Table 12: Partial accuracy results: Tessa

Algorithm	Mean Partial accuracy	Standard Deviation
MLkNN	0.65	0.029
BRkNN	0.65	0.014
Neural Network	0.76	0.018
Decision tree	0.78	0.03
Decision tree (a=0.01)	0.80	0.017
Knowledge algorithm	0.78	NA
SVM	0.80	0.014

5.8 Benchmarks

The following table reports how long it takes each algorithm to predict all 300 patients, in milliseconds. If scalability is important, e.g. a server that needs to handle a large number of requests, the load can be parallelized as the patients are independent. Additionally, it is possible to realise large speedups in the knowledge algorithm (around 100X faster) by compiling it to machine code using a tool like numba or CPython, or by rewriting it in a more performant language like Julia. This is beyond the scope of this work, but I did attempt to do this. The main challenges are that numba cannot work with standard Python dictionaries and lists, nor with Pandas dataframes – this makes it hard to write idiomatic Python code, and requires a lot of additional reworking. This would be an interesting avenue for further development.

The MLKNN algorithm is the only one that does not meet the stated criteria of less than 0.1ms per patient (i.e. less than 30ms to predict 300 patients). This is because MLKNN is computationally expensive and implemented in pure Python, whereas the sklearn algorithms mostly use C libraries like numpy behind-the-scenes.

Table 13: Benchmark results

Algorithm	Time to predict (ms)
Knowledge algorithm	2.55
Neural network	0.457
SVM	11.5
Decision tree	4.27
MLKNN	172

6 Discussion

The purpose of this section is to reflect on both the results, and the goals stated in the introduction, in order to assess whether the algorithms meet the functional and non-functional criteria. Functional criteria refers to quantitative measures like partial accuracy and time-to-predict; non-functional criteria refer to qualities like explainability and modifiability.

6.1 Which algorithm has the best performance?

Multi-label classification is inherently difficult to evaluate because there are so many performance metrics to consider. Nevertheless, a clear story emerges from the data: the knowledge-based algorithm performs better than the ML algorithms regardless of what metric one looks at, be it partial accuracy, strict accuracy, the length mismatch proportion, and precision or recall. Furthermore, the knowledge algorithm is not subject to the bias/variance tradeoff in the way ML algorithms are, since it does not overfit. Speaking strictly about performance – and ignoring any other practical considerations – the knowledge algorithm is the best. The only reason one might choose a different algorithm is if some labels are considered more important than others, because the knowledge algorithm does not perform the best for every label, even if it performs the best in the aggregate. For example, the decision tree performs better on label 10, sleep quality. But there is no reason to consider sleep quality more important than any of the other labels like sleep duration or exercise.

What surprised me, from a machine learning perspective, was how well the binary relevance algorithms (SVM, BRKNN, DT) performed compared to the multi-label algorithms (neural network and MLKNN) despite the fact that binary relevance algorithms are "blind" to the other labels. The most likely explanation for this is that the between-label interaction effects are weak. We should not be too quick in jumping to this conclusion, however, as we may wish to incorporate more recommendations in the future, leading to more complex interaction effects.

6.2 Introducing new recommendations

I chose what I thought was a reasonable number of recommendations, but this is a compromise. On the one hand, we want the algorithm to be useful to as many people as possible, therefore we would like to incorporate every relevant recommendation. On the other hand, we wanted the training set to be reasonably easy to label, and time was limited. It is very likely that we would want to add more recommendations in the future as Healthentia gains more features.

The problem with a machine learning based approach is that we would have to label an entirely new training set to take into account the new labels and input features; this is a shortcoming with any supervised algorithm. Further, the set of optimal hyperparameters might also change with the data, requiring optimisation. This is a time-consuming process. Furthermore, I remain skeptical that a binary relevance approach would continue to work well once the relationship between labels starts to grow more complex, though there is no way to know this until it is actually done. If I had to use a ML algorithm for this task, I would use a neural network despite its disadvantages, simply because it is multi-label aware. But my first choice would be the knowledge algorithm, because it is easy to adapt – new recommendations can easily be added, and inter-dependencies can be coded in. This is a major advantage in terms of time-to-MVP (Minimum Viable Product).

6.3 Explainability

Of the five different algorithms, it is fair to say that the neural network is the hardest to explain conceptually to a layperson; this is because the neural network requires a relatively deep knowledge of mathematics in order to understand. The neural network incorporates many different concepts, such as non-linear activation functions, stochastic gradient descent, linear algebra, calculus and backpropagation. Furthermore, even for an expert, it is difficult to say exactly why a neural network gives a certain recommendation, since the weights don't really have an intuitive meaning.

The other machine learning algorithms are fairly straightforward to understand and visualise. The SVM draws a line that separates each label. (Granted, this is a hyperplane, not a line, and it involves a kernel trick, but the basic concept is simple.) The decision tree can be plotted and explained split-by-split, while K-nearest neighbours is very intuitive.

I think the best algorithm in terms of explainability, however, is the knowledge algorithm. Yes, it requires a certain amount of domain knowledge to understand – but for anyone with a background in medicine, the inner workings of the algorithm would seem pretty logical.

In terms of convincing the doctors to accept eHealth technologies, a study conducted by Ketikidis et al. [43], which used the Technology Acceptance Model (TAM), showed that ease of use was the most important predictor of intention-to-use. Surprisingly, perceived usefulness was not significantly correlated with intention-to-use in this study. The authors could not explain why, and this results is not replicated by other studies. For example, a meta-analysis conducted by Strudwick et al. [44] found that both perceived usefulness and ease of use were strongly correlated with intention-to-use among nurses. Ease of use is related to UX (user experience) design and falls outside the scope of this article. Perceived usefulness, however, can be bolstered by showing high accuracy scores, and having relevant recommendations.

The issue is that the majority of studies in this area use classical models like TAM or UTAUT to evaluate the acceptance of healthcare professionals to various eHealth technologies. These are very broad, generic models that were designed to be applied to any kind of IT technology. As far as I am concerned, they do not accurately represent the unique challenges entailed by the use of machine learning in medicine. In particular, the issue of explaining black-box models has deservedly received a lot of attention, especially when there are legal ramifications [45]. In short, some of the biggest advantages of using a knowledge algorithm over a black-box algorithm like a neural network is GDPR compliance and malpractice law. Another problem is moral hazard, as discussed by Watson in the BMJ [46]. Using a knowledge-based algorithm avoids the moral hazard of diagnosing and treating patients using an algorithm that neither doctors nor patients understand.

6.4 Is domain knowledge superior to ML?

One of the interesting questions that comes up from these results is whether it even makes sense to use Machine Learning to tackle a problem that is well-understood by domain experts, when the "rules" can be easily formalised and coded in. At least for this particular problem, it would seem that indeed machine learning is not the way to go. However, there was no way to be sure of this a priori. Also, in the interest of fairness, not every kind of reasoning can be easily formalised, whereas ML algorithms tend to be pretty good at dealing with “fuzzy” reasoning.

Finally, from a product-development standpoint, many programmers don’t know much about medicine, but are probably familiar with basic machine learning techniques. Machine learning also helps to market products because it sounds sexy, though this is a rather trivial argument.

This discussion does not come up very often in the literature because researchers are focused on the latest trends in machine learning (where the funding is); expert systems are considered an old, 1980s technology. I would argue that if an expert system works as well as a machine learning algorithm, it should be the preferred solution, on account of its high explainability. Machine learning should not be the ‘default’ – particularly as eHealth adoption grows, we will see malpractice law and GDPR compliance become priorities.

6.5 Further testing

The performance of the algorithms has been judged against human coders, however, the coders are not doctors, personal trainers or nutritionists. Thus, the target labels cannot be considered the “baseline truth”. It would have been very interesting to test the performance of the algorithms against recommendations given by actual doctors, and indeed this was my intent. The main reason this was not carried out was a practical one: it is difficult to label patients using a spreadsheet, and the only realistic way to get busy professionals to label the patients would have been to carry out a proper vignette study. This would have entailed developing a fully interactive website with drop-down menus, pictures, input validation, and an SQL backend. There was not enough time to do this and carry out the machine learning. But it could definitely be done in the future.

Another obvious aspect that we have not tested is performance with real users. Since the virtual coaching algorithm is still at the early prototype stage, a pilot study would be useful. The question is, what would a pilot study with users look like? What would be the metrics by which the recommendation algorithm is judged? In my mind, there are two types of results that would be interesting. The first, user satisfaction, would incorporate a number of psychological measures like perceived relevance, sense of being personal, and whether the user would want to keep using the virtual coaching service. The second type of result would be measurement of behavioural changes. In other words, does using the virtual coach result in changes in the user’s behaviour compared to either baseline (for a longitudinal study) or control (for a cross-sectional study). It would also help to know how lasting these changes might be; the ideal study might run for several months or a few years.

6.6 Limitations

The first limitation, concerning the entire methodology, is that some of the boundaries for amber/red were decided arbitrarily (see the section titled "Labelling of the data"). Of course, I could have chosen to colour code the patients red/green – or not colour code them at all – but that was simply not practical given the volume of patients we had to label, and the experience of the coders, since we are not trained as doctors or nutritionists.

The second limitation, regarding the neural network especially, is hyperparameter optimisation. Although I have done my best to optimise the hyperparameters, I am using a rather small population for the genetic algorithm due to runtime concerns. It is not beyond the realm of possibility that the neural network could perform a bit better with a different combination of hyperparameters.

The third limitation, regarding the knowledge algorithm, is that I have implemented the knowledge algorithm using discrete rules, simply because this was the easiest way to program it. For example, if a patient eats less than 25g of sugar, the label is marked 0 in importance; if they eat more than 25g but less than 50g, the label is marked 1, and if the patient eats more than 50g, it is marked 2. But what if a patient eats 27g of sugar? One might say that the patient should make a small reduction in their consumption, but in this scenario it is likely that other labels, also marked 1, should have higher precedence. Given more time, I think a continuous scoring methodology could be developed that would perform better in these edge cases.

The fourth limitation is that the statistical power of this study is quite low because 300 patients is still a small dataset. Related to this, the fifth caveat, which has already been mentioned, is that the simulated data is not a perfect match for real data. In particular, we see that labels 2 and 4 are basically unused, but this is a consequence of how we modelled exercise, and isn't necessarily realistic. Any ML algorithm trained on the simulated dataset would perform poorly on real data where labels 2 and 4 come up more frequently.

Finally, there is a whole problem that I have not touched on: missing data and imputation. This requires its own section.

6.7 Imputation

Missing data is a problem with real-world data obtained from humans; in fact it is rare for data to be complete. There are a number of reasons why data for a patient might be missing: they may not have replied to a question/questionnaire; they may not own a smart fitness device; they may not wear the device every day, or it may have run out of battery; and so on. Neither a ML algorithm nor a knowledge-based algorithm can give a patient accurate advice if it is missing too much data.

There are two ways to handle this problem. The simplest way is to prompt the user to fill in missing data (or to charge, wear and sync their device). This is the best solution in the sense that it gives actual data, however, it is not realistic to expect the user to always respond to questions or always wear a device. An overly "pushy" approach may frustrate the user and prompt un-installation of the Healthentia app. Therefore, some sort of imputation is needed.

For the recommendation algorithm I have developed, I expect imputation will be less of a problem, since my algorithm works with the mean of time-series data. So, for example, it does not matter if the patient doesn't have sleep data for every night; we can simply take the mean of the last few weeks. However, if a variable is missing entirely, then we have a problem. It might be possible to "guess" some missing feature using available data as a predictor – so if a patient has high BMI, we can assume they eat a lot of calories, to take one example. But this is indeed a guess, and doesn't work for variables that are Missing-Completely-at-Random (MCAR). In my opinion, the best way to handle this is to prompt the user to fill in these missing input features – but allow them to continue using the virtual coach with the warning that the recommendations will be less reliable.

From a programming perspective, the easiest way to implement this would be for the knowledge algorithm (I provide a pseudocode example below; see the Appendix for actual Python implementation). For the ML algorithm, it would be more complicated, as you would need a different model to handle each type of missing input.

```
IF total_exercise is not nan:
    LABEL 1 = CALL score_exercise
ELSE:
    LABEL 1 = 0
IF total_exercise AND core_proportion are not nan:
    LABEL 2 = CALL score_core
ELSE:
    LABEL 2 = 0
```

For the virtual coaching algorithm that gives advice on a day-to-day basis, the problem is harder, and imputation will be pretty much unavoidable. Since the daily recommendation algorithm uses time-series data, time-series imputation is required, of which there are two basic types: univariate and multivariate. In the former camp, the simplest approach is to use autoregression, which is simply linear regression using the past n values; the best value of n is determined using an autocorrelation plot. If the time-series data has seasonality (for example, the patient sleeps more on weekends – which is very common), a SARIMA model might be appropriate. This model takes into account trend as well as seasonality. For irregular time series data, it may be better to use a Kalman Filter [47] or

Continuous Time Bayesian Network; that said, these types of models tend not to perform as well as dicrete-time models [48], so it would good to avoid this if possible. Rather than trying to predict values hour-by-hour, it is better to predict day-by-day. For example, it is much easier to predict if a patient will exercise on a given day than to try to predict whether they will exercise at 4 o clock in the afternoon. This would imply giving the patient recommendations once a day rather than at set hours during the day.

In any case, one should never use mean imputation. This type of imputation is very simple, but it comes with two great drawbacks: firstly, it artificially reduces the standard errors, because the imputed values have no variance. Secondly, it does not preserve the relationship between variables. [49, 50] Thirdly, for time-series data in particular, mean imputation obviously does not take into account trends or seasonality.

Multivariate time series imputation is perhaps the most complex imputation of all. One relatively simple model in this class is Vector Autoregression, which is like autoregression but where the output is a vector instead of one number, and multiple lagged variables are used to make the prediction (see [51] for more a more in-depth mathematical description). A more complex approach that might work well for our data is a Recurrent Neural Network (see the Tensorflow documentation [52]) which has the advantage of being able to take into account more complex relationships between variables, especially non-linear ones; VAR assumes linearity. As for which type of imputation would perform the best, that would have to be determined empirically. One way to do this is to use actual data that becomes available after-the-fact (for example, the patient forgot to sync their FitBit). This data could be stored in a database alongside the imputed value, allowing us to compare how closely the imputation matched real data.

For a more detailed overview, see the Research Topics included at the end of this document.

6.8 Conclusion

In terms of performance, the knowledge algorithm performed better than any of the ML algorithms: whether on partial accuracy; strict accuracy; giving the right number of recommendations; sensitivity to unbalanced data; or precision and recall, the knowledge algorithm performed the best. The knowledge algorithm also does not have the characteristic of overfitting. Furthermore, the knowledge algorithm is the most practical to implement. It is easy to modify in order to handle new features and recommendations; it is much faster than re-labelling, re-training and re-optimising the ML algorithms. This is particularly true when compared to the binary relevance algorithms, as the knowledge algorithm is inherently multi-label, and will handle interactions between labels. Another area where the knowledge algorithm is easier to work with is missing data, as it does not require creating multiple models to handle each missing input feature.

In terms of explainability, the knowledge algorithm is probably more difficult to understand than a simple ML algorithm like a decision tree or nearest-neighbours algorithm for someone with no background in medicine (in other words, patients). However, for a doctor, the knowledge algorithm would seem very logical and intuitive, more so than a ML algorithm. The knowledge algorithm is certainly easier to understand conceptually than a neural network, for both laymen and doctors. Given its huge practical advantages and superior performance, the knowledge algorithm is clearly the best solution in this particular case.

Appendix

Software Engineering

The code and data are available on Github.

I wrote 1274 lines of code in total, with a total of 796 LLOCs and 156 comments. The cyclomatic complexity of the code was analysed with `radon`. The average complexity of all the code I wrote is 3.51, which corresponds to an A grade. Cyclomatic complexity is important, because programs with a large number of branches in the logic are hard to read and difficult to debug. The main offenders in terms of cyclomatic complexity are actually type-checking statements: because Python is a dynamically-typed language, it does not enforce type constraints in the function parameters. This led to a lot of problems – some functions from the scikit-multilearn library returned sparse matrices that were incompatible with my functions. Moreover, some of my functions were deliberately not designed to work with numpy arrays but with Python lists.

It would have been possible to lower the cyclomatic complexity further by using `assert` statements instead of if statements. However, using if statements allowed me to raise exceptions, and since the cyclomatic complexity is low, I deemed this tradeoff a good one. For additional clarity, I also used Python's type annotations and docstrings.

I decided to code each algorithm in its own .py file, for several reasons. First of all, some algorithms performed better with normalisation, others with standardisation of the data. Secondly, runtime was a concern, especially with hyperparameter optimisation code – therefore, the optimal hyperparameters were stored in JSON files for easy re-use and decoupling. Finally, this approach made it much easier to digest all the data and graphs that were being generated.

However, this approach risked duplicating a lot of boilerplate code. Duplication is bad because it makes errors more likely. Therefore, I structured all of the functionality that was common between models into the helper.py file. This is the largest file in the project, with 323 lines of code.

The most complex function in the project is plot_label_accuracy_cv, which is used to generate the graphs, and its helper function cross_validate; this is because the function uses a lot of complex data structures such as lists of dictionaries and list comprehensions. One annoying characteristic of the SVM implementation is that it fails if a label only has one class. Thus, labels 2 and 4 had to be removed from the target output Y – and re-inserted in order to visualise the label accuracy graph. Inserting the missing columns in the correct place proved difficult because although numpy's insert function will accept a list of integers for the index, it does not take into account the fact that the index *changes* upon inserting the first element.

```python
def plot_label_accuracy_cv(model, X: np.ndarray, Y: np.ndarray, model_name: str, offset=[]):
    """
    Like the function above, but includes cross-validation
    :param model: An sklearn-like model
    :param X: The input features
    :param Y: The labels (must be a numpy array)
    :return: Plots a graph
    """
    list_of_cv_predictions = cross_validate(k=5, X=X, Y=Y, model=model)
    if len(offset) > 0:
        for dictionary in list_of_cv_predictions:
            values = np.zeros((dictionary['Predictions'].shape[0], len(offset)))
            # insert is a HORRIBLE function, because the index CHANGES after the first insertion
            dictionary['Predictions'] = np.insert(arr=dictionary['Predictions'],
                                                  obj=offset,
                                                  values=values,
                                                  axis=1)
            # Labels are also missing from the truth.
            dictionary['Truth'] = np.insert(arr=dictionary['Truth'],
                                                  obj=offset,
                                                  values=values,
                                                  axis=1)
    list_of_results = []
    for dictionary in list_of_cv_predictions:
        results = {}
        # Note that the MLB inverse transform is 0 indexed!
        # Deliberately hard-coded
        for i in range(0, 11):
            results[str(i)] = label_accuracy(y_true=inverse_transform(dictionary['Truth']),
                                         y_predicted=inverse_transform(dictionary['Predictions']), label=i)
        list_of_results.append(results)

    #print(list_of_results)
    final_results = {}
    final_results_stdev = {}
    # Labels are numbered 1  11  not 0  10
    # This is deliberately hard-coded
    for i in range(1, 12):
        final_results[str(i)] = mean([d[str(i-1)] for d in list_of_results])
        final_results_stdev[str(i)] = stdev([d[str(i - 1)] for d
    in list_of_results])
    #print(final_results)
    values = list(final_results.values())
    names = list(final_results.keys())
    plt.bar(names, values, color='tab:blue', yerr=list(
    final_results_stdev.values()),
            ecolor='tab:orange', capsize=3.0)
    plt.xlabel("Label number")
    plt.ylabel("Correctly predicted proportion")
    plt.title(model_name + " per-label performance")
    plt.show()

def cross_validate(k: int, X: np.ndarray, Y: np.ndarray, model) ->
    list:
    """
    This function is almost but not exactly identical to KFold
    from sklearn.
    It is intended for use by plot_label_accuracy_cv()
    :param k: The k in k-fold.
    :param X: The numpy array representing the input features
    :param Y: The numpy array representing the correct output, as
    a binary mask
    :param model: An sklearn-like model
    :return: A list of truth:prediction pairs (as dict) of length
    k
    """
    if type(Y) != np.ndarray:
        Y = Y.toarray()
    if type(k) != int:
        raise TypeError("k needs to be an integer")
    if X.shape[0] != Y.shape[0]:
        raise TypeError("Lengths need to be the same")
    if X.shape[0] % k != 0:
        raise Exception("Pick a more sensible k value. The chosen
    k value does not divide the data evenly.")
    length = X.shape[0]
    split_number = int(length / k)
    start = 0
    predictions = []
    for i in range(split_number, X.shape[0]+split_number,
    split_number):
        X_test = X[start:i]
        X_train = np.concatenate((X[i:length], X[0:start]))
        # print("Length of first part: ", X[i:length].shape[0], "
    Length of second part ", X[0:start].shape[0])
        # print("Length of X train: ", X_train.shape[0])
        # print("Length of X test: ", X_test.shape[0])
        Y_test = Y[start:i]
        Y_train = np.concatenate((Y[i:length], Y[0:start]))
```

```python
        start += split_number
        model.fit(X_train, Y_train)
        prediction = model.predict(X_test)
        prediction = prediction if type(prediction) == np.ndarray else prediction.toarray()
        predictions.append({"Predictions": prediction, "Truth": Y_test})
    assert len(predictions) == k
    return predictions
```

Missing data handling

Missing data handling in the knowledge algorithm:

```python
# Recommendations that rely on missing data cannot be given
scores[1] = self.score_exercise(input_features['total_exercise']) if not math.isnan(input_features['total_exercise']) else 0
        # Check that neither total exercise nor core proportion are nan
        scores[2] = self.score_core(total_exercise=input_features['total_exercise'],
                                   core_proportion=input_features['core_proportion']) if (math.isnan(input_features['total_exercise']) == False)
                                   and (math.isnan(input_features['core_proportion']) == False) else 0
        # And so on
```

Diet Widget

Healthentia diet widget:

- Fruits and vegetables
  - Fruit
  - Anti-oxidant fruit, e.g. berries, mangoes.
  - Vegetables
- Carbohydrates
  - Potatoes
  - Bread
    - Wholemeal bread
    - White bread

    - Rice and Grains
        * Rice
        * Grains, e.g. oats, barley, rye, porridge
    - Pasta
        * White pasta
        * Wholemeal pasta

- Dairy
    - Milk
    - Yogurt and white cheeses
    - Butter and fatty cheeses
- Protein-containing foods
    - Pulses (beans, peas, lentils)
- Fish
    - Oil-rich fish
    - Shellfish and white fish
- Eggs
- Plant-based protein (e.g. tofu)
- Meat
    - Red meat
    - White meat
    - Processed meat
- Fats
    - Nuts, seeds, oils & spreads
    - Confectionery

References

[1] U. E. Bauer, P. A. Briss, R. A. Goodman, and B. A. Bowman, “Prevention of chronic disease in the 21st century: elimination of the leading preventable causes of premature death and disability in the usa,” *The Lancet*, vol. 384, no. 9937, pp. 45–52, 2014.

[2] A. L. May, D. Freedman, B. Sherry, and H. M. Blanck, “Obesity — united states, 1999–2010,” https://www.cdc.gov/mmwr/preview/mmwrhtml/su6203a20.htm, 2013.

[3] B. Stierman, J. Afful, M. D. Carroll, T.-C. Chen, O. Davy, S. Fink, C. D. Fryar, Q. Gu, C. M. Hales, J. P. Hughes, Y. Ostchega, R. J. Storandt, and L. J. Akinbami, “National health and nutrition examination survey 2017–march 2020 prepandemic data files development of files and prevalence estimates for selected health outcomes,” https://stacks.cdc.gov/view/cdc/106273, 2021.

[4] C. S. Fox, M. J. Pencina, J. B. Meigs, R. S. Vasan, Y. S. Levitzky, and R. B. D’Agostino Sr, “Trends in the incidence of type 2 diabetes mellitus from the 1970s to the 1990s: the framingham heart study,” *Circulation*, vol. 113, no. 25, pp. 2914–2918, 2006.

[5] G. A. Bray, “Is sugar addictive?” *Diabetes*, vol. 65, no. 7, pp. 1797–1799, 2016.

[6] A. M. Jastreboff, R. Sinha, J. Arora, C. Giannini, J. Kubat, S. Malik, M. A. Van Name, N. Santoro, M. Savoye, E. J. Duran *et al.*, “Altered brain response to drinking glucose and fructose in obese adolescents,” *Diabetes*, vol. 65, no. 7, pp. 1929–1939, 2016.

[7] A. Drewnowski, D. D. Krahn, M. A. Demitrack, K. Nairn, and B. A. Gosnell, “Naloxone, an opiate blocker, reduces the consumption of sweet high-fat foods in obese and lean female binge eaters,” *The American journal of clinical nutrition*, vol. 61, no. 6, pp. 1206–1212, 1995.

[8] N. J. Weissman, “Appetite suppressants and valvular heart disease,” *The American journal of the medical sciences*, vol. 321, no. 4, pp. 285–291, 2001.

[9] W. P. T. James, I. D. Caterson, W. Coutinho, N. Finer, L. F. Van Gaal, A. P. Maggioni, C. Torp-Pedersen, A. M. Sharma, G. M. Shepherd, R. A. Rode *et al.*, “Effect of sibutramine on cardiovascular outcomes in overweight and obese subjects,” *New England Journal of Medicine*, vol. 363, no. 10, pp. 905–917, 2010.

[10] NHS, “Evidence and case studies,” https://www.england.nhs.uk/personalisedcare/evidence-and-case-studies/, 2019.

[11] ——, “Nhs prevention programme cuts chances of type 2 diabetes for thousands,” https://www.england.nhs.uk/2022/03/nhs-prevention-programme-cuts-chances-of-type-2-diabetes-for-thousands/, 2022.

[12] D. Cohen, A. Hung, E. Weinberg, and D. Zhu, “Healthtech in the fast lane: What is fueling investor excitement,” *Retrieved from McKinsey & Company: https://www. mckinsey. com/industries/pharmaceuticals-and-medical-products/our-insights/healthtech-in-the-fast-lane-what-is-fueling-investor-excitement*, 2020.

[13] T. S. Bergmo, “How to measure costs and benefits of ehealth interventions: an overview of methods and frameworks,” *Journal of medical Internet research*, vol. 17, no. 11, p. e4521, 2015.

[14] P. Lucas and L. Van Der Gaag, “Principles of expert systems,” 1991.

[15] M.-L. Zhang and Z.-H. Zhou, “Mlknn: A lazy learning approach to multi-label learning,” *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[16] D. Ten, “Multi-label classification with scikit-multilearn,” https://xang1234.github.io/multi-label/, 2018.

[17] B. Stark, C. Knahl, M. Aydin, and K. Elish, “A literature review on medicine recommender systems,” *International journal of advanced computer science and applications*, vol. 10, no. 8, 2019.

[18] N. Mahmoud and H. Elbeh, “Irs-t2d: Individualize recommendation system for type2 diabetes medication based on ontology and swrl,” in *Proceedings of the 10th International Conference on Informatics and Systems*, 2016, pp. 203–209.

[19] O. Medvedeva, T. Knox, and J. Paul, “Diatrack: web-based application for assisted decision-making in treatment of diabetes,” *Journal of Computing sciences in Colleges*, vol. 23, no. 1, pp. 154–161, 2007.

[20] A. A. Hamed, R. Roose, M. Branicki, and A. Rubin, “T-recs: Time-aware twitter-based drug recommender system,” in *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 2012, pp. 1027–1031.

[21] A. Yazdanyar and A. B. Newman, “The burden of cardiovascular disease in the elderly: morbidity, mortality, and costs,” *Clinics in geriatric medicine*, vol. 25, no. 4, pp. 563–577, 2009.

[22] M. R. Islam, I. Shafique, K. Rahman, and A. Haque, “A simple study on weight and height of students,” *European Scientific Journal*, vol. 13, no. 6, pp. 63–71, 2017.

[23] N. M. Punjabi, “The epidemiology of adult obstructive sleep apnea,” *Proceedings of the American Thoracic Society*, vol. 5, no. 2, pp. 136–143, 2008.

[24] F. Rosqvist, D. Iggman, J. Kullberg, J. Cedernaes, H.-E. Johansson, A. Larsson, L. Johansson, H. Ahlström, P. Arner, I. Dahlman *et al.*, “Overfeeding polyunsaturated and saturated fat causes distinct effects on liver and visceral fat accumulation in humans,” *Diabetes*, vol. 63, no. 7, pp. 2356–2368, 2014.

[25] Y. Li, A. Hruby, A. M. Bernstein, S. H. Ley, D. D. Wang, S. E. Chiuve, L. Sampson, K. M. Rexrode, E. B. Rimm, W. C. Willett *et al.*, “Saturated fats compared with unsaturated fats and sources of carbohydrates in relation to risk of coronary heart disease: a prospective cohort study,” *Journal of the American College of Cardiology*, vol. 66, no. 14, pp. 1538–1548, 2015.

[26] T. M. Barber, S. Kabisch, A. F. Pfeiffer, and M. O. Weickert, “The health benefits of dietary fibre,” *Nutrients*, vol. 12, no. 10, p. 3209, 2020.

[27] L. A. Schmidt, “New unsweetened truths about sugar,” *JAMA internal medicine*, vol. 174, no. 4, pp. 525–526, 2014.

[28] C. M. Champagne, H. Han, S. Bajpeyi, J. Rood, W. D. Johnson, C. J. Lammi-Keefe, J.-P. Flatt, and G. A. Bray, “Day-to-day variation in food intake and energy expenditure in healthy women: the dietitian ii study,” *Journal of the Academy of Nutrition and Dietetics*, vol. 113, no. 11, pp. 1532–1538, 2013.

[29] J. Gianoudis, C. Bailey, and R. Daly, “Associations between sedentary behaviour and body composition, muscle function and sarcopenia in community-dwelling older adults,” *Osteoporosis international*, vol. 26, no. 2, pp. 571–579, 2015.

[30] M. Brown, “Strength training and aging,” *Topics in Geriatric Rehabilitation*, vol. 15, no. 3, pp. 1–5, 2000.

[31] J. Frija-Masson, J. Mullaert, E. Vidal-Petiot, N. Pons-Kerjean, M. Flamant, M.-P. d’Ortho *et al.*, “Accuracy of smart scales on weight and body composition: observational study,” *JMIR mHealth and uHealth*, vol. 9, no. 4, p. e22487, 2021.

[32] M. Kolatt, "Game of thrones' the mountain lost 50 kilos while eating 4000 calories per day: here is his exact diet and workout routine," https://www.t3.com/features/game-of-thrones-star-drops-an-insane-110-lbs50kg-while-eating-almost-4000-calories-a-day, 2021.

[33] Z. Villines and D. Bubnis, "Body fat percentage charts for men and women," https://www.medicalnewstoday.com/articles/body-fat-percentage-chart, 2020.

[34] L. J. Appel, J. M. Clark, H.-C. Yeh, N.-Y. Wang, J. W. Coughlin, G. Daumit, E. R. Miller III, A. Dalcin, G. J. Jerome, S. Geller *et al.*, "Comparative effectiveness of weight-loss interventions in clinical practice," *New England Journal of Medicine*, vol. 365, no. 21, pp. 1959–1968, 2011.

[35] M. A. Matteson, "Effects of a cognitive behavioral approach and positive reinforcement on exercise for older adults," *Educational Gerontology: An International Quarterly*, vol. 15, no. 5, pp. 497–513, 1989.

[36] A. N. Dalton and S. A. Spiller, "Too much of a good thing: The benefits of implementation intentions depend on the number of goals," *Journal of Consumer Research*, vol. 39, no. 3, pp. 600–614, 2012.

[37] S. S. Iyengar and M. R. Lepper, "When choice is demotivating: Can one desire too much of a good thing?" *Journal of personality and social psychology*, vol. 79, no. 6, p. 995, 2000.

[38] A. Bilogur, "Multi-label classification with neural networks," https://www.kaggle.com/residentmario/multi-label-classification-with-neural-networks, 2019.

[39] K. Pykes, "The vanishing/exploding gradient problem in deep neural networks," https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11, 2020.

[40] V. Bushaev, "Stochastic gradient descent with momentum," https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d, December 2017.

[41] N. S. Chauhan, "Hyperparameter optimization for machine learning models," https://www.kdnuggets.com/2020/05/hyperparameter-optimization-machine-learning-models.html, May 2020.

[42] M. S. Sorower, "A literature survey on algorithms for multi-label learning."

[43] P. Ketikidis, T. Dimitrovski, L. Lazuras, and P. A. Bath, "Acceptance of health information technology in health professionals: an application of the revised technology acceptance model," *Health informatics journal*, vol. 18, no. 2, pp. 124–134, 2012.

[44] G. Strudwick, "Predicting nurses' use of healthcare technology using the technology acceptance model: an integrative review," *CIN: Computers, Informatics, Nursing*, vol. 33, no. 5, pp. 189–198, 2015.

[45] A. M. Froomkin, I. Kerr, and J. Pineau, "When ais outperform doctors: confronting the challenges of a tort-induced over-reliance on machine learning," *Ariz. L. Rev.*, vol. 61, p. 33, 2019.

[46] D. S. Watson, J. Krutzinna, I. N. Bruce, C. E. Griffiths, I. B. McInnes, M. R. Barnes, and L. Floridi, "Clinical applications of machine learning algorithms: beyond the black box," *Bmj*, vol. 364, 2019.

[47] I. Trading, "The kalman filter for financial time series," https://www.r-bloggers.com/2010/05/the-kalman-filter-for-financial-time-series/, 2010.

[48] M. Liu, F. Stella, A. Hommersom, P. J. Lucas, L. Boer, and E. Bischoff, "A comparison between discrete and continuous time bayesian networks in learning from clinical time series data with irregularity," *Artificial intelligence in medicine*, vol. 95, pp. 104–117, 2019.

[49] K. Grace-Martin, "Missing data: Two big problems with mean imputation," https://www.theanalysisfactor.com/mean-imputation/, 2020.

[50] D. Radecic, "Stop using mean to fill missing data," https://towardsdatascience.com/stop-using-mean-to-fill-missing-data-678c0d396e22, 2019.

[51] J. H. Stock and M. W. Watson, *Introduction to econometrics, updated.* Pearson Education, 2015, https://www.econometrics-with-r.org/16-1-vector-autoregressions.html.

[52] F. Chollet, "Time series forecasting," https://www.tensorflow.org/tutorials/structured_data/time_series#recurrent_neural_network, 2022.

Research Topics

Alex Bujorianu

September 2, 2022

1 Abstract

A medical recommendation algorithm that advises patients on lifestyle changes, based on data collected by the Healthentia platform, is proposed. The problem of giving recommendations is a treated as a multi-label classification, where a patient may receive 1–3 pieces of advice out of a set of possible suggestions. The research would compare a supervised learning approach (decision tree, neural network, and KNN) with a handcrafted rule-based algorithm; the quality of the different approaches will be evaluated against the recommendations of a medical professional, using the Hamming Loss and Kripendorff's alpha statistics. The problem of imputation is discussed and the MLE approach is chosen. For time-series imputation, a Continuous Time Bayesian Network is considered along with Kalman filter. The scope of the evaluation is limited to just medical accuracy, as following the patients long-term is not possible.

2 Introduction

Thesis objective: The goal of my thesis research is to design and implement a recommendation algorithm that gives general health & wellbeing advice to Healthentia patients. Healthentia is a mobile app developed by Innovation Sprint which is used to monitor patient data (collected mainly by wearables) and to aid doctors conducting clinical trials. The algorithm is intended for use by Healthentia users with a wide variety of different illnesses, comorbidities, and behaviours, and focuses mainly on lifestyle changes; but it should also be possible to modify the algorithm to accommodate different kinds of patients who require more specialised advice, such as patients with COPD (Chronic Obstructive Pulmonary Disease).

Based on this, I formulate a number of different research questions.

Research Question 1: According to the literature, what are the best features for a virtual coaching medical recommendation algorithm, and how does this compare to the data that is present in the Healthentia platform? This is an empirical question, but is also necessary to inform the design science of a recommendation algorithm.

Research Question 2: According to the literature, what is the best methodology for designing an *interpretable, adjustable, personal* and medically accurate health & wellbeing recommendation algorithm? By interpretable, I mean that the algorithm is not a "black box": it can be explained and understood by humans. Adjustable here means that the algorithm can be easily tailored towards groups of patients with specific illnesses that require specialist advice. Personal means that the algorithm adapts and responds to the individual user's inputs. Medical accuracy is self-explanatory: the algorithm must provide good advice, or at least, it must not hand out *dangerous* advice.

Research Question 3: How should missing data be handled by the recommendation system at runtime, and during training? Given that much of the available data relies on user action – be it self-reported information about diet and exercise, adherence to charging and wearing a smart device, etc. – it is likely that the data will be incomplete, or split into irregular time intervals. A few different approaches are possible: there are various methods of imputation, and some machine learning algorithms are more robust to missing data than others. The chosen solution must be sound from a statistical point of view: it most not underestimate standard errors, and it must preserve the relationship between variables.

Research Question 4: How should the medical accuracy of a recommendation algorithm be evaluated? Should the algorithm be tested according to its agreement with experts, or should it be evaluated based on user opinions?

Research Question 5: How should persuasion principles be incorporated in an electronic health intervention system in order to produce positive changes in lifestyle and behaviour? This question is about translating theory from the psychological literature into actionable advice.

2.1 What is a recommendation?

So far, I have discussed the term recommendation without giving a concrete example of what constitutes a recommendation. In my mind, a good recommendation has a three key properties:

- it is actionable;
- it is personalised rather than generic;
- and naturally, it must be medically correct.

Let's discuss each property in more depth.

Actionable By actionable, I mean that the recommendation is not vague, but focuses on one concrete action that the patient can make. Thus, "eat healthily" is not a good recommendation because it is non-specific; a better recommendation would be something along the lines of, "I notice that you eat a lot of foods high in saturated fat, like cheese, butter and pork. These foods substantially increase your LDL (bad) cholesterol and cause many cardiovascular complications. Maybe you can try eating more unsaturated fats like those found in nuts, olive oil and avocados?"

Personalised Notice, furthermore, that this recommendation is personal: "I notice that **you** eat a lot of..." Personalised recommendations are more effective than one-size-fits-all imperatives that are addressed to whomever it may concern (indeed the entire modern advertising industry is based on this principle). Personalisation is done on the basis of data we have about the patient, such as their dietary habits, medical conditions etc.

Medically correct Finally, the recommendation is medically accurate because it reflects decades of research into the link between saturated fats in the diet and cardiovascular disease. [1, 2, 3] as well as AHA guidelines. In general, recommendations must be supported by high-quality RCT (randomised clinical trial) or intervention-based research that can establish cause-and-effect; note that I used the word **cause** and not "associated with" (a perennial favourite in nutrition research backed by statistical analysis of survey data). Recommendations must also follow best practices as set out by the relevant medical experts and public health authorities, such as the European Heart Network, American Heart Association and so on.

3 Available Data

The following data is available in the Healthentia platform. Note that some variables are only available for patients with a certain diagnosis, for example, diabetics have blood sugar data whereas other patients do not.

1. Age.
2. Sex.
3. Body Mass Index: reliable, easy to calculate, but quite a crude metric. BMI does not tell you the body fat percentage or where the fat is located in the body: is it visceral or subcutaneous fat? Is it stored in the belly or in the thighs, arms and bum?
4. Sleep patterns: duration, quality of sleep (nightmares, interruptions), and stages (N1, N2, N3, REM). Also, time taken to fall asleep. Is this self-reported or measured by a device?
5. Level of activity: number of steps taken (as measured by a smart wearable) and self-reported data.
6. Pre-existing conditions diagnosed by a medical professional.
7. Diet: self-reported. Studies indicate that self-reported data collected by food questionnaires is unreliable, as people don't remember what they ate, and obese people

drastically under-report the amount they eat. [4, 5, 6] Therefore, the veracity of the data needs to be taken into account.

8. Heart rate: resting heart rate and peak measured by a wearable in beats per minute.

9. Cholesterol, particularly LDL vs HDL.

10. (For diabetics) Glucose readings over time. We need to distinguish between fasting glucose vs peak glucose after a meal.

11. (For people with heart disease) Systolic and diastolic blood pressure (measured in mmHg).

12. Possibly, test results, which can contain various data like inflammatory markers IL-1, TNF and lactate dehydrogenase; antibody titers; hemoglobin; white blood cell count; vitamin and mineral readings like vitamin D and calcium. This data is ad-hoc, so whether it can actually be used for a recommendation algorithm is debatable. On the other hand, if it is available, it's potentially very useful, high-quality data.

All of this data is **unlabelled**: there is no recommendation(s) associated with a patient. That said, the dataset can be considered partially labelled on the basis that some of the variables are easily to classify; for example, glucose and cholesterol have well-standardised normal ranges. Some variables may be a little harder to classify: to judge whether the resting heart rate is within a healthy range, we need some information regarding what activity the patient is doing at the time of the reading. We can combine this data with other available information, such as location (we can determine whether the patient is at the gym or not, cycling, etc.)

4 Possible Approaches

4.1 Recommendation Algorithms

This section addresses research question 2, concerning how best to design a medical recommendation algorithm. To avoid confusion, I should clarify that when I say recommendation *algorithm*, I really mean recommendation *system*. The recommender system will likely use multiple "helper algorithms" to pre-process the data, to perform classification, and impute missing values. There are two basic types of machine learning algorithms: supervised and unsupervised. A supervised approach requires some of the data to be labelled to make a training set, whereas an unsupervised algorithm does not need labels. For our purposes, we need to use a supervised algorithm and create a labelled training set.

The way I will approach this problem is to treat it as a multi-label classification problem: we have a patient with multiple features (activity level, blood sugar, sleep, phenotype, diagnosis etc.) and we wish to choose the best advice out of a set of possible recommendations.

In some cases, we may want to hand out more than one recommendation – as they are not mutually exclusive – which makes this problem multi-label (in multiclass classification it is assumed the categories are mutually exclusive). We may also decline to offer any advice until the patient provides more information, as the strength of the evidence is insufficient to draw a conclusion. Therefore, we would ideally prefer a probabilistic model over a discriminative model, as probabilistic models give an indication of the strength of a classification. If two recommendation "classes" have similar probabilities, we can choose both.

A neural network with a sigmoid activation function in the output layer is the standard solution for a multi-label neural network classifier. The softmax activation function should not be used because it assumes the classes are mutually exclusive—this assumption makes the softmax function better at optimising exclusive classification problems, but works against it in a multi-label problem. For example, if two neurons both have the value 0 (i.e. neither class is a good recommendation) a neural network with a softmax activation function will pick a winner anyway. In cases where multiple classes are correct, using softmax will split the confidence between them. [7] For the hidden layers, either ReLU or sigmoid activation functions would work, and which is better would have to be determined empirically. The number of neurons in the output layer will equal the number of possible recommendations.

Since one of the conditions stipulated in RQ1 is that the algorithm must be understandable, such a neural network should be as simple as possible, while still providing good representational power. It is helpful if the neural network does not have too many hidden layers (i.e. it is not very deep). Crucially, neural networks can deal with a missing input value, by coding it as 0. An input neuron with a value of 0 will not have an impact on the output of the other neurons in the hidden layer, since the other neurons calculate a dot product between the input vector and the weight vector. If we, for example, have a patient with no blood glucose data present – because they are not diabetic – this means the recommendation algorithm is unlikely to give inappropriate advice that is meant for diabetics. It also means we can use the same algorithm for patients with different diseases. In practical terms, I would most likely use the `MLPClassifier` in the scikit-learn library. The scikit documentation recommends using the `lbfgs` solver for small training sets.

Another approach would be to use a Decision Tree algorithm. This has the benefit of being more easily understood than the sometimes opaque neural network, as it is a simple algorithm with an intuitive interpretation: the input data is split on the features in order to separate classes. (I will leave out the exact details of how to achieve an optimal split; which heuristic to use, be it classification error rate, Gini impurity or entropy; whether to prune the tree and how etc. These are implementation details.) Additionally, modern implementations of decision trees, like CART, can deal with missing data. However, there are some issues with decision trees that might not necessarily make them the best for this task. Firstly, they work best with discrete features rather than continuous variables (ratio data) like blood glucose level in nmol/litre, which have to be discretised. A good portion of our data is continuous so this is not ideal. Secondly, decision trees are discriminative models, so they are not good at giving estimating uncertainty. Thirdly, decision trees, like many machine learning models, have a tendency towards overfitting. This is discussed in the section below.

K nearest neighbours (KNN) can also be used to solve a multi-label classification problem using the `MLkNN` implementation in Scikit. The main advantage of this model is that, although it is not a generative model, it gives us a better idea of the "fuzziness" of a prediction than a typical discriminative model. Because KNN calculates the distance of the data point to the nearest classes, we can tell which label was a "near miss".

4.2 Overfitting

Overfitting is a common phenomenon in machine learning whereby a model with high expressive power—that is, a model capable of learning many patterns in the data—will start to learn the noise in the data. Thus, the model loses its ability to generalise well on unseen data. It is less likely to be a problem with models that have a lower expressive power, such as Naive Bayes, and quite likely to happen with neural networks, particularly if they have many hidden layers (deep networks). For neural networks, L1 or L2 normalisation is used to reduce overfitting. This adds a penalty term to the loss function that penalises large weights.

Decision trees are a "greedy" algorithm and are known to overfit. There are various solutions, such as using random forest, and pruning the tree based on various heuristics like information gain. It is also possible to use the chi squared statistic for pruning, but this does not work well if the training set is small, as even good splits would be statistically insignificant. This is important because we know our training set will be quite small.

The standard approach for evaluating overfitting is to split the labelled set into train, test and validation sets. The performance of the model is evaluated on both the testing and validation set: if the accuracy improves in the testing set but starts to drop off in the validation set, we know that the model is overfitting. Since our labelled set is small, however, this approach is inappropriate because it gives an unrealistic idea of the model's performance for both bias (underfitting) and variance (overfitting). The standard solution for dealing with small datasets is to use k-fold cross-validation, with a k value of 10 being the rule-of-thumb. [8] The idea is to split the data into mini-sets (randomly shuffling the data between them), testing the model on one set and training it on the other sets, and doing this repeatedly for all the different combinations in order to calculate the mean performance, and variance.

4.3 Approaches to Multi-Label Classification

Multi-label classification presents some challenges that multi-class classification does not. To begin with, the search space equals 2^N, so the search space increases exponentially with the number of labels [9]. We would like to keep the number of labels, i.e. recommendations, small because of this, but we also want the recommendations to be useful for every patient, so there is a trade-off there. I think somewhere around 10–30 labels would be manageable.

It is possible to use unsupervised algorithms to cluster related recommendations together. One such example is the `NetworkXLabelGraphClusterer` (ibid). This is typically done when there are many labels. To understand clustering intuitively, think about comorbidities. Patients who are obese typically have diabetes and cardiovascular problems, and will likely receive similar recommendations. So we expect "eat fewer saturated fats" and "limit the intake of foods with a high Glycemic Load" (for example) to be clustered together. Using an unsupervised algorithm to first cluster the labels can help the supervised algorithm to perform better.

It is possible to treat each target variable in the multi-label classification problem separately, and combine the results. This method is known as Binary Relevance (ibid). The main issue with this method is that is slow if the label space is large, but should be manageable if the number of labels is not too high. The scikit multilearn library contains a good practical implementation of this solution, and it works well with a decision tree.

Another approach is to use Label Powerset. This reduces the multi-label classification problem into a multi-class classification problem, where each target variable is combined and each combination is treated as a class (ibid). This method requires all possible labels to be in the training set.

The main advantage of a neural network compared to a decision tree is that it is inherently multilabel, and does not need to use either of the above strategies. A neural network gives a probability for each label, and the top 3 can be selected, or only one if the probabilities are low.

4.4 How big should the training set be?

The main problem with any supervised algorithm, particularly a neural network, is that it requires a sufficiently large training set to learn from, whereas all of our data is unlabelled. I can think of two ways to bridge this problem: one, to look for a dataset containing recommendations for patients. Since this field of study is quite new, it may be difficult to find such a dataset, particularly if the input features differ from the ones that are available. The second solution would be to find some experts who can help me label the data. This seems viable, but realistically, such a dataset would end up being quite small – at most 100 patients, possibly as low as 20 or 30. A training set of n=100 would be large enough, but twenty or thirty labels would be insufficient to train a neural network properly: if we consider the perceptron learning rule,

$$w^{new} = w^{old} - \eta \Delta E(w) \tag{1}$$

We see that we can increase the learning rate η to make the algorithm converge faster. However, this risks making the algorithm "overshoot" (i.e. not converge to the most optimal value). Moreover, it would be very difficult to validate this algorithm to ensure it has a good bias-variance tradeoff; it seems unlikely the algorithm would generalise well to a larger population, as it cannot learn more subtle patterns in the data from such a small training set.

4.5 A Knowledge-Based System

Perhaps the best approach, then, and the one I discussed with my company supervisor, is to just do what doctors do: apply a series of rules based on domain knowledge. The advantage of this approach is that it does not require a training set, and it is also extremely understandable: it can be visualised with a simple flowchart, and experts in the field will have no trouble understanding the reasoning. Thus, the recommendation algorithm can be thought of as a kind of "virtual GP". It should be noted that this approach is equivalent to a decision tree, except that the splits are constructed manually rather than by a machine. Such manual algorithms are sometimes referred to as "expert systems" or "knowledge-based systems" in the literature. [10]

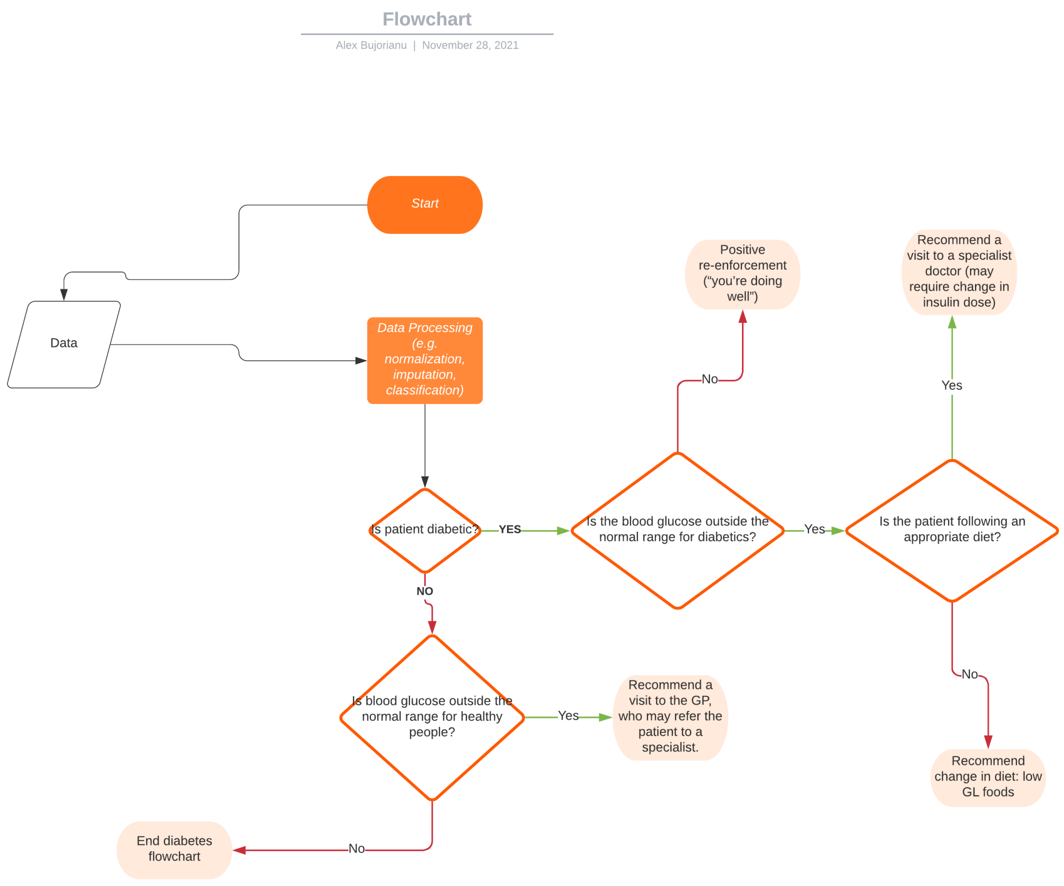
Consider the simple example provided in figure 1.

Let's go through the reasoning. Firstly, we want to know if the patient is classed as diabetic: this can be a Boolean variable. If the patient is diabetic, we look at whether their glucose value falls within the acceptable values range for diabetics. If it does, we provide positive re-enforcement for the patient, so that they feel motivated to continue with their good habits. If not, we look at the data we have about diet, which we have classified as appropriate for diabetics, or not. If we know the diet is not appropriate – the most likely explanation is that the patient is eating too many refined carbohydrates such as white bread and pasta, rice etc. – we tell the patient to change their diet by eating foods with a lower GI/GL (Glycemic Index/Glycemic Load), such as fruit instead of desserts, wholemeal grains, beans and legumes, as well as foods with more fat, like walnuts and olive oil. If we believe the patient is eating correctly, then we recommend a visit to a specialist doctor, who may adjust their insulin dose, or prescribe medication to regulate glucose levels.

If the patient is not classed as diabetic but has abnormally high glucose values, we recommend they visit the doctor, as they may be an un-diagnosed diabetic. We could also use proxy values instead of blood glucose, as people not classed as diabetics typically don't measure their glucose levels: for example, we can ask them if their urine smells sweet.

Note that the flowchart simplifies the real algorithm for the sake of illustration. To begin with, flowcharts are not very good at modelling time, and they assume a fixed process, whereas this algorithm changes and adapts over time based on past inputs (in technical terms, it is not memoryless). Secondly, the decision rules seem quite simple, which they are for the sake of illustration; in reality, a decision rule would involve time-series and statistical analysis. So we would not just look at whether the blood glucose is high *now* – but we would look at whether it is *consistently* high over time.

Some pseudocode is provided in figure 2.



Flowchart
Alex Bujorianu | November 28, 2021
Start
Data
Data Processing (e.g. normalization, imputation, classification)
Is patient diabetic?
YES
NO
Is the blood glucose outside the normal range for diabetics?
Yes
No
Positive re-enforcement ("you're doing well")
Is the patient following an appropriate diet?
Yes
Recommend a visit to a specialist doctor (may require change in insulin dose)
No
Recommend change in diet: low GL foods
Is blood glucose outside the normal range for healthy people?
Yes
Recommend a visit to the GP, who may refer the patient to a specialist.
No
End diabetes flowchart

Figure 1: A flowchart showing the decision logic for patients who are diabetic, or are not classed as diabetic, but have high blood sugar.

4.6 Prioritising the Recommendations

We have formalised the problem of giving recommendations as being a multi-label classification problem with between 1–3 recommendations per patient. The limit of 3 recommendations is for practical reasons, to keep it manageable for patients. If we were to use a knowledge-based system, we could easily end up giving more than 3 recommendations though, so this has to be taken into account in the control flow. We need to score the recommendations and give only the most important and relevant advice.

This can be done using a scoring system, taking into account data about the patient and combining it with domain knowledge. This could be on a scale of 1–3, with 3 indicating high importance (e.g. very high blood sugar levels, large number of cigarettes smoked per day, very little sleep etc.) and 1 indicating low importance. A score of 0 would indicate irrelevance. The recommendation and its importance could be stored as a key-value pair (in a tuple or a dictionary), inserted into a list, and the list can be sorted based on the importance score. The first three recommendations are then selected.

4.7 Time Series Analysis

As we have seen in the pseudocode example, we need to perform some time series analysis. Mainly, we're interested in detecting changing trends over time, such as changes in exercise or diet. We want to provide positive re-enforcement to a patient if the data shows they are making a lifestyle change. The length of the moving average window is important: we have to smooth out minor day-to-day fluctuations. This would have to be determined empirically, but a 3 day window seems like a reasonable guess. Most time-series models such as ARIMA (Autoregressive integrated moving average) and Kalman Filter deal with moving averages, which are fine in many cases, but possibly insufficient in the case of diabetes. For a diabetic patient, the mean is less important; we want to see a reduction in hypoglycaemia events (blood glucose levels below 70 mg/dl) as well as a reduction in hyperglycaemia (above 140 mg/dl).

Using the output of the time-series model, we apply a classification to the patient, such as "Decreasing trend", "Increasing trend" or "Staying the same" next to blood sugar levels or saturated fat intake.

5 Imputation

5.1 Training Set Imputation

Data with high missingness is a frequent problem in data analysis, and there are two main ways to approach the problem. One, to subset only the data for which observations are available. This approach is impractical when there are multiple variables, and can significantly limit the size of the available training set. At runtime, missing data is usually unavoidable. The second approach is to use imputation; however, the method of imputation needs to be carefully considered in order to ensure it has good statistical properties. Mean imputation is the simplest method, but it is considered statistical malpractice. Martin [11] shows that mean imputation introduces two problems: firstly, it artificially reduces standard errors, because the imputed data does not have any variance. Secondly, it does not preserve the relationship between variables – for example, it can weaken a strong correlation, or artificially inflate it. Two methods have been developed to achieve a better quality of imputation: Maximum Likelihood (MLE) and MICE (Multivariate imputation with chained equations). Firstly, I will discuss some assumptions that apply when doing multivariate imputation, regardless of the method.

5.2 Multivariate imputation

When doing imputation with multiple variables, it is important to consider whether relationships between the variables should be taken into account. For example, if a patient is missing LDL cholesterol measurements but does have high blood pressure, we have to take into account that people with high blood pressure tend to have high LDL cholesterol.

There are three types of assumptions that can be in multivariate imputation: Missing-Completely-at-Random (MCAR), Missing-at-Random (MAR) and Not-Missing-at-Random (NMAR). The least realistic assumption is MCAR, where we assume that the variables are independent. In equation 2 consider response, R, a 0/1 encoded variable; X is a vector without missing values, and Y is a vector with some missing values [12]. If missing values are MCAR, then the missing values have nothing to do with the observed values in Y.

$$P(R=1|X,Y) = P(R=1) \tag{2}$$

MAR is a more realistic assumption. It assumes that the variables can be correlated, like in the example with LDL cholesterol and blood pressure. It does still assume that patients with missing data are missing the data for random reasons (such as forgetting to the charge the battery) and not because of bias, such as sicker patients not using the device. Formally, MAR states that the probability of the response depends on another variable for which we have observations. See 3.

$$P(R=1|X,Y) = P(R=1|X) \tag{3}$$

For data that is NMAR (Not Missing at Random), it is extremely difficult to make a good imputation solution [12]. But this should not be a problem in our case.

5.3 MICE and MLE

The first, more sophisticated method of imputation is MICE. This has the advantage of theoretical robustness [13] and has a practical implementation in Python [14]. In particular, MICE takes into account the fact that other variables can be predictive of the missing data. However it is very complicated and hard to understand. What MICE does is generate multiple datasets with slightly different imputations. This is typically done with a linear regression model and a random draw from a standard normal distribution. If we have 3 variables, Z, X and Y, of which Z has missing values and X & Y are complete, then the linear regression equation will look something like equation 4 – see [12].

$$\hat{Z} = b_0 + b_1 X + b_2 Y + sE \tag{4}$$

(Where s is the root mean square error (RMS) and E is a draw from a standard normal.)

From equation 4 we see that MICE makes two assumptions: that missing values can be predicted with a linear model (i.e there is a linear relationship between the variable with missing values and the complete predictor variables), and that the errors follow a multivariate normal distribution. One obviously has to be careful with these assumptions. If there are categorical or binary variables in the dataset, FCS (Fully Conditional Specification) can be used instead of MCMC (Markov Chain Monte Carlo), as it uses logistic regression for the categorical or binary variables.

Both FCS and MCMC are similar in that they create multiple datasets with imputed values, and update the weights of the linear regression model (b) iteratively – this makes them somewhat similar to a machine learning algorithm. The downside is that this process is very memory-intensive, especially for large datasets or many iterations. The iterations are required in order to converge to a good posterior distribution and ensure that the imputed values between datasets are statistically independent.

The second approach I will consider is imputation by Maximum Likelihood Estimation. In general, the idea behind MLE which is to find a parameter θ that maximises the likelihood function, which is typically done by taking the log-likelihood function, as finding the derivative of a sum is easier than a product. If the log-likelihood function is monotonically increasing, the solution that maximises the log-likelihood also maximises the likelihood function. This is not always the case, so this assumption needs to be treated with some caution. Generally, MLE works well when the underlying data has a normal distribution or some other common parametric distribution.

Note that this parameter θ is *not* the parameter to some theoretical distribution; it is, rather, the missing value which we are trying to find. Consider the case that the the first variable y_1 is missing a value at the ith observation. If you look at equation 5, you will see that we sum the joint pdf function f of all the other variables, over the observations of y_1 for which we have data. (If the variables are continuous, we use integrals instead of summations, but the basic idea is the same.)

$$L = \prod_{1}^{n} \sum_{y_1} f(y_{i1} \ldots y_{ik}, \theta) \tag{5}$$

Allison [12] argues that MLE is a superior approach compared to MICE, for several reasons:

1. Allison points out that MICE always gives a different result (because of randomisation) whereas MLE does not. This is an issue for reproducibility, but it can easily be worked around by setting a seed value for the random number generator.
2. MLE is more memory efficient than MICE, especially for large datasets. Allison does not mention this, but it could be a problem.
3. Allison points out that MICE requires making many more decisions than MLE, which introduces uncertainty.
4. If the relationship between missing data and predictor variables is non-linear, then obviously one cannot use a MICE algorithm that depends on linear regression. The assumption of normality is another assumption that MICE makes and MLE does not.
5. Finally, in MICE, the analysis model must be compatible with the imputation model. In particular, the analysis model must not include variables that were not included in the imputation model. Likewise, if one is using a non-linear or interaction model, e.g. a neural network, one also has to use an imputation model that takes into account these interactions and non-linearities, adding further complexity.

The variety of issues associated with MICE makes me lean towards MLE as the superior solution.

One might reasonably wonder if a neural network could be used for imputation and not just regression. This is an area of active research: Yoon [15] used a Generative Adversarial Net with good results, and Cheng [16] had success using a deep-learning (multiple hidden layers) neural network. See [17] for a discussion about the merits of this imputation approach. As promising as it looks, there aren't – to the best of my knowledge – any off-the-shelf machine learning algorithms for imputation yet.

5.4 Imputation at Runtime

So far I have discussed imputation in the training set. There are some key differences between imputing data on a training set versus at runtime. For one, it is possible to avoid having to impute data at all in a training set (although this may not be desirable, especially if it makes the training set small). During runtime, imputation is likely to be unavoidable. However, it is possible to prompt the user to input some missing data, especially if that data is very important to the final outcome. It would be nice to have some way of judging the importance of a variable to an outcome; one simple way to do this is using Principal Component Analysis (PCA). The top 3 variables with the highest explained variance can be prioritised (for example). In terms of the control flow, the user should be prompted to input the important missing data before any recommendations are calculated.

5.5 Time Series Imputation

Performing imputation on time-series data is quite different from performing imputation on non-time-series data. To begin with, time-series data is not i.i.d (identically distributed and independent): there is a strong correlation between my cholesterol levels today and my cholesterol levels yesterday, for example. Secondly, the time-series data is likely to be **irregular**. A diabetic, for example, does not measure their blood sugar levels every 24 hours like clockwork, but in irregularly spaced intervals. Additionally, it is possible that the data for one variable might not depend purely on past values for that variables (univariate time-series) but that it might have dependencies on other variables for which we have observations (multi-variate time-series). For example, a diabetic's blood sugar levels also depend on their activity level and their diet. In other words, univariate imputation implicitly assumes MCAR.

A good option would be to use a Continuous Time Bayesian Network (CTBN). Research indicates that when the time-series data is irregularly spaced, MAR rather than MCAR, and there is no strong prior knowledge about the distribution of the data, continuous time Bayesian networks perform equally or better than discrete time Bayesian networks. [18] Another approach would be to use a Generative Adversarial Network (GAN) but this typically requires a large training set, which would be problematic. [19] The main issue with using a CTBN is that it requires a Directed Acrylic Graph (DAG), which is a model-based representation of the causality between different variables. This often requires the help of a domain expert to construct. For a CTBN, we can use MLE parameter estimation, which makes no particular assumptions about the underlying prior probability, or a Bayesian estimator. MLE has a tendency to overfit on a small datasets, whereas a Bayesian estimator is more theoretically robust. The Bayesian Dirichlet equivalent uniform prior distribution can used as a reasonable starting point. [20] In terms of a practical implementation, `pomegranate` cannot be used because it only implements a discrete Bayesian network. One promising library would be `tsBNgen` .

Alternatively, we can choose to use a simpler method by assuming the time-series data is MCAR, that is, it does not depend on other variables (for which we have data). Although this assumption is a little unrealistic, a multivariate model might not necessarily perform better than a univariate model if we have few observations for the other variables, or the dependent variables are only weak predictors. Applying the principle of Occam's Razor, we should prefer a simpler model if that's the case. One good candidate is the Kalman Filter, which is available in the `imputets` R package. Basically, the Kalman Filter calculates a moving average of the data [21]. So if we notice that a patient's exercise level has been trending up for a good period of time, we can take this into account for imputing a day that has no data. The Kalman filter is good at dealing with data that is somewhat noisy and sampled at irregular time intervals.

6 Persuasion Principles

This section addresses research question 5. The purpose of the Healthentia virtual coaching platform is ultimately to produce a beneficial change in lifestyle and behaviour. But to do this, we need to be aware of human psychology and incorporate this *a priori* knowledge in the design of our recommendations. Consider the following New Year's resolution: "This year, I will go to the gym and lose weight." We've likely all heard some variation of this promise (the reader may even have made it themselves). But how often is it successful? The person typically buys a gym membership, goes there for a week, maybe a month, and then gradually loses interest. Research shows that between 50–60% of New Year resolutions fail. [22, 23]

So how can we improve on this? Both Norcross and Oscarsson found that "approach-orientated" goals are more effective than avoidance-orientated goals. An approach-orientated goal is one in which the patient tries to introduce a positive habit, such as going to the gym or learning to play the ukulele. An avoidance-orientated goal is about quitting a bad habit. For example: "This year, I will give up on smoking." Oscarsson found that the group which received **some support** had a significantly higher success rate than the group which received no support: the 95% confidence interval was 59–65% for the support group versus 53–58% for the control group. This is important, because the Healthentia virtual coach can act like a human personal trainer, providing support and smart goal-setting.

It is known that certain bad habits are physiologically addictive; nicotine is recognised as being extremely addictive. Research has shown it is comparable to cocaine in addictiveness. [24] The fact that nicotine or caffeine is addictive is uncontroversial, but could unhealthy eating habits be motivated by similar mechanisms? In other words, is sugary or fatty food addictive? An article published in Diabetes [25] tries to answer this very question. The author adduces a fascinating study by [26] which found that, in obese adolescents, consumption of fructose or glucose was associated with an increase in activity in the limbic system (the "pleasure centre" of the brain) and a decrease in the activity of the pre-frontal cortex, which is responsible for executive control. This did **not** occur in lean, healthy adolescents. Furthermore, consumption of fructose and glucose lowers the levels of acyl-ghrelin, effectively increasing hunger (ibid). A study by Drewnowski [27] found that blocking opioid receptors using naxolone resulted in a decrease in binge-eating behaviour. Whether sugar can be considered addictive in the pharmacological sense is debatable, and beyond the purpose of this work to answer. But we have to recognise the fact that sugary foods have profound effects on the brain and metabolic system, and that this pattern of behaviour is therefore very difficult to break.

Dieting has been shown to be more effective in conjunction with pharmacological interventions [28]. Previous appetite-suppressing drugs, such as fenfluramine, were withdrawn from the market due to their association with cardiac abnormalities. [29] Until as recently as 2010, sibutramine was available on the market, and was withdrawn after a randomised clinical trial found a significant association with sibutramine use and heart attacks. [30] Appetite suppressants are an ongoing area of research, but so far, there do not appear to be many medications available that are both safe enough to be used without medical supervision, and effective.

Finally, I wish to draw on the research that has been conducted in the field of electronic persuasion systems, including advertising and social media. Perhaps the most well-known author in this space is Robert Cialdini [31]. He came up with six "persuasion principles", after which this section is named. They are:

1. Reciprocity: When humans receive an unsolicited gift, they feel obligated to return the favour. This does not necessarily have to be in the form of financial gifts; it can be a small thing, like a personalised gift card. The reader may be inclined to ask if it is practical to include this principle in a piece of software (because how can software offer a physical gift?) But virtual gifts can be used as well. Reddit, for example, has implemented the ability to give virtual gifts ("Awards") to other users.

2. Consistency: What Cialdini seems to mean by this word is that humans are inclined to honour public commitments. Subtly changing the formulation of a question or suggestion, so that the user makes a promise, can significantly improve commitment to a particular task. For example, "*Will you* please call if you change your plans?" (pause) is more effective than "Please call if you change your plans." The former prompts the user to make a commitment. The Healthentia Virtual Coach could employ a similar principle, asking the user for input at key times.

3. Social Validation: Humans have a subconscious idea that the more people believe in or comply with an idea, the more valid that idea is. This explains the success of religion, for example, or bestsellers. This principle is one of the most frequently used in marketing, and one hardly needs any imagination to see how it could be incorporated into Healthentia recommendations. They practically write themselves: "Millions love their oats for breakfast, as it is filling, fibre-rich, and nutritious. Try it now." Cialdini does give a very clear warning to health authorities who try to use this principle in the wrong way, as it can backfire to produce the opposite effect of what is desired. Crucially, this principle cannot be used to discourage bad behaviour. In one egregious example, when a New Jersey suicide prevention programme informed teenagers of the high number of teenage suicides, the kids actually become more likely to consider suicide a viable solution, found David Shaffer and his colleagues at Columbia University. [1]

4. Liking: People are more likely to be persuaded by a person they like, be it that the person is attractive, praises them, or fights for their cause. The most practical way to

[1]Cialdini mentions this example in the 2001 Scientific American article cited, but I was unable to find a citation for this specific claim. The best source I could find is a 1991 study by Shaffer et al., but the article is paywalled. [32]

incorporate this in the Healthentia Virtual Coach would be to use praise or positive re-enforcement.

5. Authority: This one is straightforward. People are more likely to trust in recommendations made by experts, so recommendations could start, "According to the American Heart Association, eating soluble fibre reduces your chance of getting a heart attack."

6. Scarcity: If people perceive something to be scarce, or the *knowledge* about that product being scarce, they are more likely to buy it. This is one principle that I don't think is relevant for our purposes.

Of the six, I think authority and social validation are the most important for a medical recommendation algorithm, for they are both powerful and easy to implement. Liking and consistency could also be employed, with some creativity. Reciprocity and scarcity are the least relevant.

But can we test whether the formulation of a recommendation makes a difference to the patient's response? It would be possible to perform randomised A/B testing on the patients to test this hypothesis. The patients could be asked to rate the recommendation with a thumbs up/thumbs down, or on a Likert scale of usefulness. The main theoretical problem with this approach is that patient *satisfaction* is not necessarily a good proxy for recommendation effectiveness in the long-run; that a patient is happy with the advice does not mean they act on it. But it would at least give an indication of the perceived relevance of the recommendations. The main practical problem is whether there would be sufficient time to incorporate the algorithm into the Healthentia app in order to carry out A/B testing.

7 Evaluation

This section concerns RQ4, evaluation of the medical accuracy of the algorithm. There are two types of evaluation that are useful for this algorithm: 1) an evaluation of whether a recommendation is medically appropriate for a patient, and 2) an evaluation of whether these recommendations actually lead to a change in lifestyle. The first problem relates directly to Research Question 4. The second evaluation is unfortunately not possible to do, as it would require following the patients for an extended period of time (months, years) to see if their weight improves, or if they exercise more, and so on.

My proposed method for evaluating the medical accuracy of the recommendations is to to use expert evaluation: one or more medical experts can offer 1–3 recommendations per patient to a group of patients, out of a set of possible recommendations, based on the available data. As discussed in the multi-label classification section, I expect we will have anywhere from 10–30 possible recommendations, which should be manageable for the doctors. If there are multiple experts, we ask them to advise different patients (since we are not interested in the agreement between experts, but in the agreement between experts and the machine.) The experts labeling the test set would have to be different people from the expert who helped me label the training set, to prevent bias. The output of the algorithm is then compared to the experts' recommendations, and an agreement score can be calculated on the nominal ratings.

Calculating agreement scores between the algorithm and experts is tricky, because this is a multi-label classification and not an exclusive classification. This involves some challenges: it is no longer the case that an agreement is hard wrong/right, but rather, we are interested in the number of labels that the algorithm got right, e.g. 2 out of 3 or 1 out of 3 (we find have to find what is common between two sets). It also means that traditional tests of agreement, for example Cohen's kappa, Bhapkar or McNemar, cannot be used. [33].

One possibility is to use Kripendorff's alpha. This statistic was developed in response to the need for an inter-coder reliability metric in qualitative research [34]. It has several nice properties that make it ideal for this problem: it can handle multi-label data; it can handle missing data; and it works with a wide variety of data, be it nominal, ordinal or ratio. Finally, Kripendorff's alpha is also very easy to understand—a score of 1 indicates perfect agreement, and 0 indicates no agreement. A practical implementation can be found on pypi.

It is also possible to use the same evaluation metrics for multi-label classification as for exclusive classification, such as precision, recall, and F1 score, with the use of micro-averaging or macro-averaging. In micro-averaging, we average the individual true positive, false positives and false negatives. [35] In macro-averaging, we average the precision and recall on the different labels (ibid.) Macro-averaging is useful for judging performance at-a-glance, and micro-averaging gives a more in-depth performance overview. Micro-averaging is particularly important when the data is **unbalanced**, i.e. some labels are recommended frequently while others are rarely recommended; it allows us to judge whether the algorithm is good at predicting the less frequent recommendations, which might otherwise be overshadowed by its average performance.

For accuracy scores, we can use Hamming Loss, which tells us the fraction of labels that were predicted correctly (ibid). A much stricter accuracy score would be exact match, which as the name suggests, only classifies the sets of labels that match exactly as being correct. For our purposes, Hamming Loss would be a better metric, because it is unlikely we will have a perfect match, especially with so many labels.

One might wonder if it is possible to refute the null hypothesis that the recommendations of the algorithm agree with the experts' recommendations by chance. This would be a problem if we only had, say, 5 possible recommendations from which to pick the 3. This is the problem that Cohen's kappa aims to solve. However, the standard Cohen's kappa cannot be applied to a multi-label classification problem, although extensions that would allow for this have been suggested, e.g. using weighting [36]. But since we will have anywhere from 10 to 30 possible labels, it seems unlikely that the results would agree by chance. I am also not sure if any implementations of multi-label Cohen's kappa are available.

8 List of Abbreviations

- BMI: Body Mass Index
- COPD: Chronic Obstructive Pulmonary Disease
- FCS: Fully Conditional Specification, a type of algorithm used for multivariate imputation by chained equations.
- IL-1: Interleukin 1, an inflammatory marker.
- HDL: High-Density Lipoprotein, a type of cholesterol that has a protective effect on the heart.
- KNN: K-nearest neighbours, an algorithm that performs classification based on how far a point is from the other classes.
- LDL: Low-Density Lipoprotein, a type of cholesterol that causes blood vessels to narrow and clog, leading to heart attacks.
- MAR: Missing at Random. This assumes that data is missing at random, but the missing values can depend on other observed variables. For example, high blood pressure is correlated to high LDL cholesterol.
- MCAR: Missing Completely at Random: A generally unrealistic scenario where missing values are assumed to be independent of of other observed variables.
- MICE: Multivariate Imputation by Chained Equations.
- MCMC: Markov Chain Monte Carlo, another type of MICE algorithm.
- MLE: Maximum Likelihood Estimation. A way of estimating population parameters from a sample.
- MOM: Method of Moments. A way of calculating population parameters based on moments, like the mean.
- MSE: Mean Square Error. A measure of how well an estimate fits the population parameter.
- REM: Rapid Eye Movement sleep.
- RCT: Randomised Controlled Trial, a type of clinical study where one group is given a placebo, and the other group receives the intervention. Patients are assigned to one group at random, to prevent bias. An RCT can be single-blind, where the doctor knows what group the patient is in, or double-blinded, where neither patient nor doctor know, only the study investigators.

- RMS: Root Mean Square Error.
- TNF: Tissue Necrosis Factor, a biomarker of inflammation.

References

[1] F. Rosqvist, D. Iggman, J. Kullberg, J. Cedernaes, H.-E. Johansson, A. Larsson, L. Johansson, H. Ahlström, P. Arner, I. Dahlman *et al.*, "Overfeeding polyunsaturated and saturated fat causes distinct effects on liver and visceral fat accumulation in humans," *Diabetes*, vol. 63, no. 7, pp. 2356–2368, 2014.

[2] K. Kupferschmidt, "Scientists fix errors in controversial paper about saturated fats," 2014. [Online]. Available: https://www.science.org/content/article/scientists-fix-errors-controversial-paper-about-saturated-fats

[3] Y. Li, A. Hruby, A. M. Bernstein, S. H. Ley, D. D. Wang, S. E. Chiuve, L. Sampson, K. M. Rexrode, E. B. Rimm, W. C. Willett *et al.*, "Saturated fats compared with unsaturated fats and sources of carbohydrates in relation to risk of coronary heart disease: a prospective cohort study," *Journal of the American College of Cardiology*, vol. 66, no. 14, pp. 1538–1548, 2015.

[4] J. I. Macdiarmid and J. Blundell, "Dietary under-reporting: what people say about recording their food intake," *European journal of clinical nutrition*, vol. 51, no. 3, pp. 199–200, 1997.

[5] D. A. Schoeller, "How accurate is self-reported dietary energy intake?" *Nutrition reviews*, vol. 48, no. 10, pp. 373–379, 1990.

[6] M. Livingstone, A. Prentice, J. Strain, W. Coward, A. Black, M. Barker, P. McKenna, and R. Whitehead, "Accuracy of weighed dietary records in studies of diet and health." *British Medical Journal*, vol. 300, no. 6726, pp. 708–712, 1990.

[7] A. Bilogur, "Multi-label classification with neural networks," 2019. [Online]. Available: https://www.kaggle.com/residentmario/multi-label-classification-with-neural-networks

[8] J. Brownlee, "A gentle introduction to k-fold cross-validation," 2018. [Online]. Available: https://machinelearningmastery.com/k-fold-cross-validation/

[9] D. Ten, "Multi-label classification with scikit-multilearn," 2018. [Online]. Available: https://xang1234.github.io/multi-label/

[10] P. Lucas and L. Van Der Gaag, "Principles of expert systems," 1991.

[11] K. Grace-Martin, "Missing data: Two big problems with mean imputation," 2020. [Online]. Available: https://www.theanalysisfactor.com/mean-imputation/

[12] P. D. Allison, “Handling missing data by maximum likelihood,” in *SAS global forum*, vol. 2012, no. 312, 2012, pp. 1038–21.

[13] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, “Multiple imputation by chained equations: what is it and how does it work?” *International journal of methods in psychiatric research*, vol. 20, no. 1, pp. 40–49, 2011.

[14] D. Radecic, “Stop using mean to fill missing data,” 2019. [Online]. Available: https://towardsdatascience.com/stop-using-mean-to-fill-missing-data-678c0d396e22

[15] J. Yoon, J. Jordon, and M. Schaar, “Gain: Missing data imputation using generative adversarial nets,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 5689–5698.

[16] C.-Y. Cheng, W.-L. Tseng, C.-F. Chang, C.-H. Chang, and S. S.-F. Gau, “A deep learning approach for missing data imputation of rating scales assessing attention-deficit hyperactivity disorder,” *Frontiers in psychiatry*, vol. 11, p. 673, 2020.

[17] M. Smieja, Ł. Struski, J. Tabor, B. Zieliński, and P. Spurek, “Processing of missing data by neural networks,” *arXiv preprint arXiv:1805.07405*, 2018.

[18] M. Liu, F. Stella, A. Hommersom, P. J. Lucas, L. Boer, and E. Bischoff, “A comparison between discrete and continuous time bayesian networks in learning from clinical time series data with irregularity,” *Artificial intelligence in medicine*, vol. 95, pp. 104–117, 2019.

[19] M. Tadayon, “A python library to generate a synthetic time series data,” 2020. [Online]. Available: https://towardsdatascience.com/tsbngen-a-python-library-to-generate-time-series-data-from-an-arbitrary-dynamic-bayesian-network-4b46e1

[20] L. M. Amorosa, “A bayesian network to model the influence of energy consumption on greenhouse gases in italy,” 2020. [Online]. Available: https://pgmpy.org/detailed_notebooks

[21] I. Trading, “The kalman filter for financial time series,” 2010. [Online]. Available: https://www.r-bloggers.com/2010/05/the-kalman-filter-for-financial-time-series/

[22] J. C. Norcross, A. C. Ratzin, and D. Payne, “Ringing in the new year: The change processes and reported outcomes of resolutions,” *Addictive Behaviors*, vol. 14, no. 2, pp. 205–212, 1989.

[23] M. Oscarsson, P. Carlbring, G. Andersson, and A. Rozental, “A large-scale experiment on new year’s resolutions: Approach-oriented goals are more successful than avoidance-oriented goals,” *PLoS One*, vol. 15, no. 12, p. e0234097, 2020.

[24] J. E. Henningfield, C. Cohen, and J. D. Slade, "Is nicotine more addictive than cocaine?" *British journal of addiction*, vol. 86, no. 5, pp. 565–569, 1991.

[25] G. A. Bray, "Is sugar addictive?" *Diabetes*, vol. 65, no. 7, pp. 1797–1799, 2016.

[26] A. M. Jastreboff, R. Sinha, J. Arora, C. Giannini, J. Kubat, S. Malik, M. A. Van Name, N. Santoro, M. Savoye, E. J. Duran *et al.*, "Altered brain response to drinking glucose and fructose in obese adolescents," *Diabetes*, vol. 65, no. 7, pp. 1929–1939, 2016.

[27] A. Drewnowski, D. D. Krahn, M. A. Demitrack, K. Nairn, and B. A. Gosnell, "Naloxone, an opiate blocker, reduces the consumption of sweet high-fat foods in obese and lean female binge eaters," *The American journal of clinical nutrition*, vol. 61, no. 6, pp. 1206–1212, 1995.

[28] B. Brody, "Appetite suppressants: What you should know," 2020. [Online]. Available: https://www.webmd.com/diet/appetite-suppressants

[29] N. J. Weissman, "Appetite suppressants and valvular heart disease," *The American journal of the medical sciences*, vol. 321, no. 4, pp. 285–291, 2001.

[30] W. P. T. James, I. D. Caterson, W. Coutinho, N. Finer, L. F. Van Gaal, A. P. Maggioni, C. Torp-Pedersen, A. M. Sharma, G. M. Shepherd, R. A. Rode *et al.*, "Effect of sibutramine on cardiovascular outcomes in overweight and obese subjects," *New England Journal of Medicine*, vol. 363, no. 10, pp. 905–917, 2010.

[31] R. B. Cialdini, "The science of persuasion," *Scientific American*, vol. 284, no. 2, pp. 76–81, 2001.

[32] D. Shaffer, A. Garland, V. Vieland, M. Underwood, and C. Busner, "The impact of curriculum-based suicide prevention programs for teenagers," *Journal of the American Academy of Child & Adolescent Psychiatry*, vol. 30, no. 4, pp. 588–596, 1991.

[33] J. Uebersax, "Statistical methods for diagnostic agreement," 2015. [Online]. Available: http://john-uebersax.com/stat/agree.htm

[34] A. F. Hayes and K. Krippendorff, "Answering the call for a standard reliability measure for coding data," *Communication methods and measures*, vol. 1, no. 1, pp. 77–89, 2007.

[35] K. Nooney, "Deep dive into multi-label classification," 2018. [Online]. Available: https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

[36] A. Rosenberg and E. Binkowski, "Augmenting the kappa statistic to determine interannotator reliability for multiply labeled data points," in *Proceedings of HLT-NAACL 2004: Short Papers*. Boston, Massachusetts, USA: Association for Computational Linguistics, May 2 - May 7 2004, pp. 77–80. [Online]. Available: https://aclanthology.org/N04-4020

```
READ PatientData
#See if one of the top 3 variables has a high missingness score
IF IS.NA PatientData["Important_variable1", "Important_variable2", "Important_variable3"] > 0.4
    PRINT("Sorry, we do not have sufficient data to provide a recommendation. Please can you fill in the following missing data.")
    PROMPT_INPUT
ELSE
    FILL_MISSING PatientData
#Classify_glucose() calls some time series functions.
DEFINE Classify_Glucose(PatientData):
    IF time_series_peaks(PatientData["Glucose"]) > 140
        PatientData["Glucose_classification] = "Too high"
    ELIF time_series_lows(PatientData["Glucose"]) < 70:
        PatientData["Glucose_classification] = "Too low"
    ELSE:
        PatientData["Glucose_classification] = "Just right"
CALL Classify_Glucose(PatientData)
IF PatientData["Diagnosis"] IS "Diabetic"
    GO TO diabetes
ELSE
    IF PatientData["Glucose_classification"] IS "Too high" OR "Too low"
        RECOMMEND("Your blood glucose levels are outside the normal levels for a healthy person. We recommend a visit to the GP.")
    ELSE
        #Go to some other subroutine
SUBROUTINE diabetes:
IF PatientData["Glucose_classification"] IS "Just right"
    RECOMMEND("You're keeping your diabetes under control. Well done!")
ELSE
    IF PatientData["Diet"] IS "Appropriate"
        RECOMMEND("Please see your endocrinologist, you may need an adjustment to your medication or insulin dose.")
    ELSE
        RECOMMEND("Hmm, it seems you're having some trouble keeping your diabetes under check. Diabetics should follow a diet with a low glycemic load (GL). You should try to eat more wholegrain foods (e.g. oats, beans, lentils, wholegrain bread and pasta) and fewer sugary foods and processed carbohydrates like white bread, rice and crisps.")
```

Figure 2: Pseudocode for flowchart 1.